NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A273 206

DTIC
ELECTE
DEC 0 1 1993
A

THESIS

OBJECT RECOGNITION THROUGH IMAGE
UNDERSTANDING FOR AN AUTONOMOUS
MOBILE ROBOT

by

Mark J DeClue

September 1993

Thesis Advisor   Yutaka Kanayama

Approved for public release; distribution is unlimited.

93-29353

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1993	3. REPORT TYPE AND DATES COVERED Master's Thesis	

4. TITLE AND SUBTITLE
Object Recognition Through Image Understanding for an Autonomous Mobile Robot

5. FUNDING NUMBERS

6. AUTHOR(S)
DeClue, Mark Joseph

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-500

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public use; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The problem addressed in this research was to provide a capability for sensing previously unknown rectilinear, polyhedral-shaped objects in the operating environment of the autonomous mobile robot *Yamabico-11*. The approach to the system design was based on the application of edge extraction and least squares line fitting algorithms of [PET92] to real-time camera images with subsequent filtering based on the environmental model of [STE92]. The output of this processing was employed in the recognition of obstacles and the determination of object range and dimensions. These measurements were then used in path tracking commands, supported by *Yamabico's* Model-based Mobile Robot Language (MML), for performing smooth, safe obstacle avoidance maneuvers. This work resulted in a system able to localize objects in images taken from the robot, provide location and size data, and cause proper path adjustments. Accuracies on the order of one to ten centimeters in range and one-half to two centimeters in dimensions were achieved.

14. SUBJECT TERMS
mobile robotics, computer vision, edge extraction, obstacle avoidance

15. NUMBER OF PAGES
198

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Object Recognition Through Image Understanding
For An
Autonomous Mobile Robot

by
Mark J. DeClue
Lieutenant, United States Navy
B. S. in Computer Science, United States Naval Academy, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
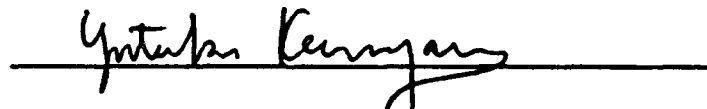
from the

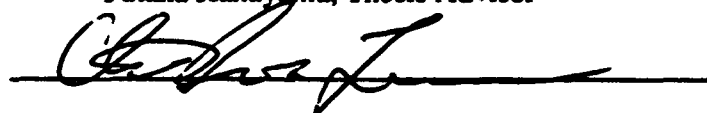**NAVAL POSTGRADUATE SCHOOL**

September 1993

Author: _____

Mark Joseph DeClue

Approved By: _____

Yutaka Kanayama, Thesis Advisor

_____

Chin-Hwa Lee, Second Reader

_____

Ted Lewis , Chairman,
Department of Computer Science

ABSTRACT

The problem addressed in this research was to provide a capability for sensing previously unknown rectilinear, polyhedral-shaped objects in the operating environment of the autonomous mobile robot *Yamabico-11*. The approach to the system design was based on the application of edge extraction and least squares line fitting algorithms of [PET92] to real-time camera images with subsequent filtering based on the environmental model of [STE92]. The output of this processing was employed in the recognition of obstacles and the determination of object range and dimensions. These measurements were then used in path tracking commands, supported by *Yamabico's* Model-based Mobile Robot Language (MML), for performing smooth, safe obstacle avoidance maneuvers. This work resulted in a system able to localize objects in images taken from the robot, provide location and size data, and cause proper path adjustments. Accuracies on the order of one to ten centimeters in range and one-half to two centimeters in dimensions were achieved.

DTIC QUALITY INSPECTED 5

Accesion For		
NTIS CRA&I		☑
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distibution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

iii

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Object recognition is a common application of computer vision and in fact has become a common component in many manufacturing processes requiring recognition of specific parts and the subsequent transport to various locations. In much of the research in this area, either a search is conducted for a specific object within the confines of the image or heuristics are applied in analyzing the image in an attempt to identify objects. A specific branch of computer vision, robot vision, has been the subject of significant research in the past decade. While many of the basic approaches have undergone progressive refinement, numerous new directions continue to be pursued in both general and system specific applications. A vision system for an autonomous vehicle may be employed for a variety of uses including navigation, object recognition, and environmental mapping. Of course, the usefulness of any such system is limited by the robotic platform into which it is integrated. In previous research at the Naval Postgraduate School, robot position determination through camera vision was addressed in [PET 92] and resulted in accuracies varying from a few inches to more than one foot. However, recent work using line segment extraction via sonar [MAC 93] on the *Yamabico-11* robot has provided positioning accuracies on the order of a fraction of a centimeter, virtually eliminating the usefulness of implementing the visual system as originally intended.

B. OVERVIEW

Although *Yamabico* may have precise knowledge of its location in a given environment, it is only capable of detecting the presence of unexpected obstacles in its path when relying on sonar as the sole sensor. The ability to determine position accurately does allow a new approach to be evaluated which incorporates vision as an additional sensor with complementary capabilities to that of sonar. Specifically, if a model of the

environment and its location within that environment are known, it should be possible to immediately detect unexpected elements in what the robot camera sees by filtering out those elements which are in a generated image based on the model. This alleviates the need for pre-processing the image before any interpretation may be attempted. This makes isolation of an obstacle fairly straight forward. Hence, the fusion of object recognition through vision and the current precise locomotion capability can provide the necessary foundation for intelligent and autonomous obstacle avoidance. It is on this basic premise that the vision system for *Yamabico* has been designed and implemented.

II. PROBLEM STATEMENT

The problem being addressed in this work is as follows:

Given a mobile autonomous vehicle with accurate knowledge of its position in a partially known environment, develop the capability to detect objects in a video image taken from the robotic platform by recognizing elements unknown to the environmental model, and subsequently extract range and dimension data which will aid in avoidance maneuvers. This information should be derived solely from the information provided by the vision system without input from other active external sensing systems.

A. ASSUMPTIONS

The following assumptions were incorporated into the research in order to focus the work on the image understanding problem and ensure the goals were realizable with a single camera system (i.e. no stereo vision).

1. Orthogonal World

The robot is operating in a partially known, orthogonal world. The world is partially known in that the robot has information on the location of fixed walls, but has no information on the positions of doors or the presence of obstacles in the world. The world is orthogonal in that all walls meet at right angles. The normal operating environment for *Yamabico* is the fifth floor of Spanagel Hall, which adequately meets the above assumptions.

2. Obstacle Constraints

An obstacle is any object that is not part of the known world. The placement of obstacles is arbitrary, with a restriction that they rest on the floor and not be suspended in the air. This is not unreasonable since it would be highly unlikely for a suspended obstacle to be encountered which is not a permanent part of the operating environment. An additional constraint on the research is that when an object is encountered, it is assumed

3

that it is positioned orthogonal to the robot so that base edges can be used in calculating ranges with a single camera. Obstacles are assumed to be stationary while the robot is in the vicinity of the obstacle. However, if the robot leaves and then returns to the same area, the obstacle may be in the same position, a different position, or it may be gone.

3. Environment Model and Localization

The experimentation assumed the existence of a three-dimensional, wire-frame model of the robot's operating environment as well as accurate knowledge of the position within this domain. These requirements are certainly realizable, and in fact each has been achieved in previous work at the Naval Postgraduate School.

B. APPROACH

Like many research projects, the job of employing vision in support of obstacle avoidance for an autonomous mobile robot entails a step-wise progression toward the desired capability.

1. Object Extraction

The fundamental problem is to derive the ability to recognize the existence of an obstacle in the robot's path which could prevent it from safely carrying out its assigned task or proceeding to the desired goal. This basic functionality requires knowledge of what the robot anticipates encountering combined with subsequent recognition of elements in the environment which conflict with this a priori expectation. Requirements inherent in this task are proper generation of the expected view (provided by an environmental model in the case of this work), processing of an input video image, pattern matching between the two, and a subsequent localization of an object. The localization process can become more difficult when multiple objects are considered. The problem at this stage is to develop an efficient algorithm which can properly group associated line segments together to form each object. Once this has been accomplished, the same approach as was used for the single object case can be applied to obtain individual object dimension and location data.

4

2. Evaluation

Once the presence of an "object" has been confirmed, it is desirable to evaluate the dimensions and location of the object so that appropriate avoidance measures may be pursued. At this stage, the vision system is capable of providing information which will allow initial path corrections to be carried out but can not provide insight into how long the object will be of concern to the robot. This limitation exists because the data obtained from the image is based solely on two-dimensional information, and the classification of the object type is not generally required in order to adequately carry out this fundamental assessment. It should be noted that this aspect of the research is actually the central concern, with a primary goal of ensuring robust and consistent implementation of this capability.

3. Obstacle Avoidance

One of the most notable aspects of *Yamabico* is a fine and extensive locomotion capability, making obstacle avoidance relatively straight forward. Combining the range and dimension data generated in the evaluation described above with the variety of path types available through the Model-based Mobile Robot Language (MML) used to control the robot enables the performance of a smooth, safe obstacle avoidance maneuver. Specifically, the use of parabolic-curved elements would enable the robot to accurately track a path specified according to Voroinoi boundaries between the obstacle and the surrounding environment, thus providing the maximum safe path [KAN93]. Since the motion is based on a path-tracking approach, the robot can automatically make smooth transitions between newly computed paths.

4. Object Identification

Although not pursued in this work, the ability to extract enough information from the image for identification of the object would provide three-dimensional information, thus enhancing the "intelligence" of the robot's obstacle avoidance maneuvers. This topic is in itself a widely researched field and numerous approaches have been suggested for a wide range of applications. One technique which appears to be especially attractive for

implementation on *Yamabico* is the alignment method [ULL91]. The basic premise of this approach is that given a known set of feature points for a known object and the same points on an unknown object, it is possible to map the two sets via constant coefficient linear equations if they are alike. The powerful aspect of this relationship is the fact that it is valid regardless of rotational and/or translational differences, permitting direct analysis of image objects according to the object database. Once the object has been classified, available information would include the object depth, which provides the final parameter needed to carry out complete avoidance measures solely on the basis of visual input.

III. *YAMABICO-11* ROBOT

Yamabico-11, shown in Figure 3.1, is an autonomous mobile robot used as a test platform for research in path planning, obstacle avoidance, environment exploration, path tracking, and image understanding.

A. HARDWARE

The robot is powered by two 12-volt batteries and is driven on two wheels by DC motors which drive and steer the robot while four spring-loaded caster wheels provide balance.

The master processor is a MC68020 32-bit microprocessor accompanied by a MC68881 floating point co-processor. (This is the exact same CPU as a Sun-3 workstation). This processor has one megabyte of main memory and runs with a clock speed of 16MHz on a VME bus. An upgrade to a SPARC-4 processor with 16 megabytes of main memory was nearing completion when this research was completed.

All programs on the robot are developed using a Sun 3/60 workstation and UNIX operating system. These programs are first compiled and then downloaded to the robot via a RS-232 link at 9600 baud rate using a PowerBook™ 145 computer for the communication interface. The new SPARC-4 board will be accessible via an ethernet connection, decreasing download time from about five minutes to a few seconds.

Twelve 40 kHz ultrasonic sensors are provided as the primary means by which the robot senses its environment. The sonar subsystem is controlled by an 8748 micro-controller. Each sonar reading cycle takes approximately 24 milliseconds.

The visual images from the robot are generated by a JVC TK870U CCD camera head equipped with a FUJINON TV zoom lens. The unit is mounted along the centerline of the robot at a height of 34 inches. This camera provides a NTSC standard RGB video image through a video framer attached to a Silicon Graphics Personal Iris™ workstation as shown in Figure 3.2. The framer digitizes the sync and composite signals for storage on the Iris

7
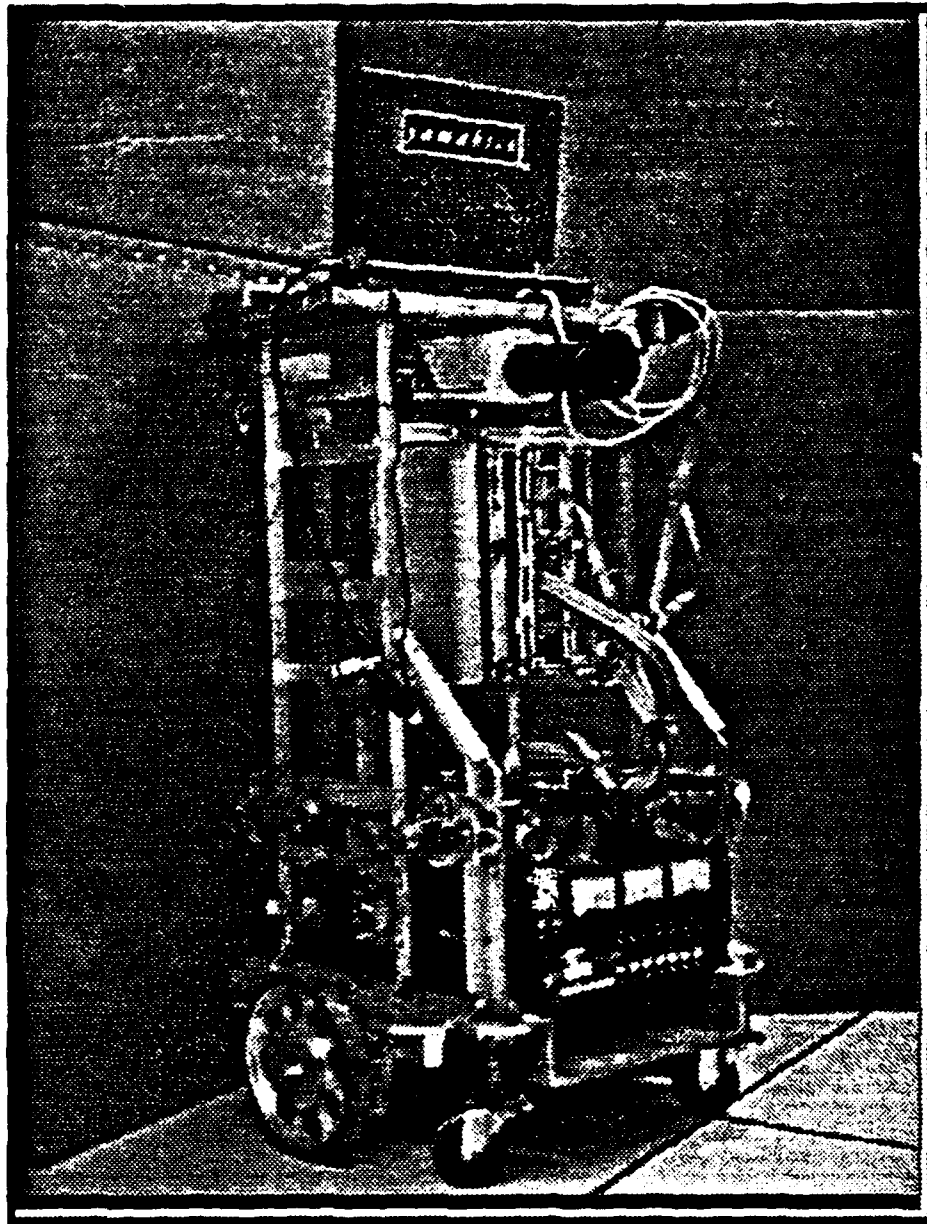
Figure 3.1: *Yamabico-11* Mobile Robot

and also passes the signal to a high definition monitor. Currently, the video signal is transmitted via standard coaxial video cable to the image processing hardware. When

Composite & Sync
Video Cables

CAMERA

SGI   VIDEO   FRAMER

SGI

PERSONAL

IRIS

DIAMOND

VISION

MONITOR
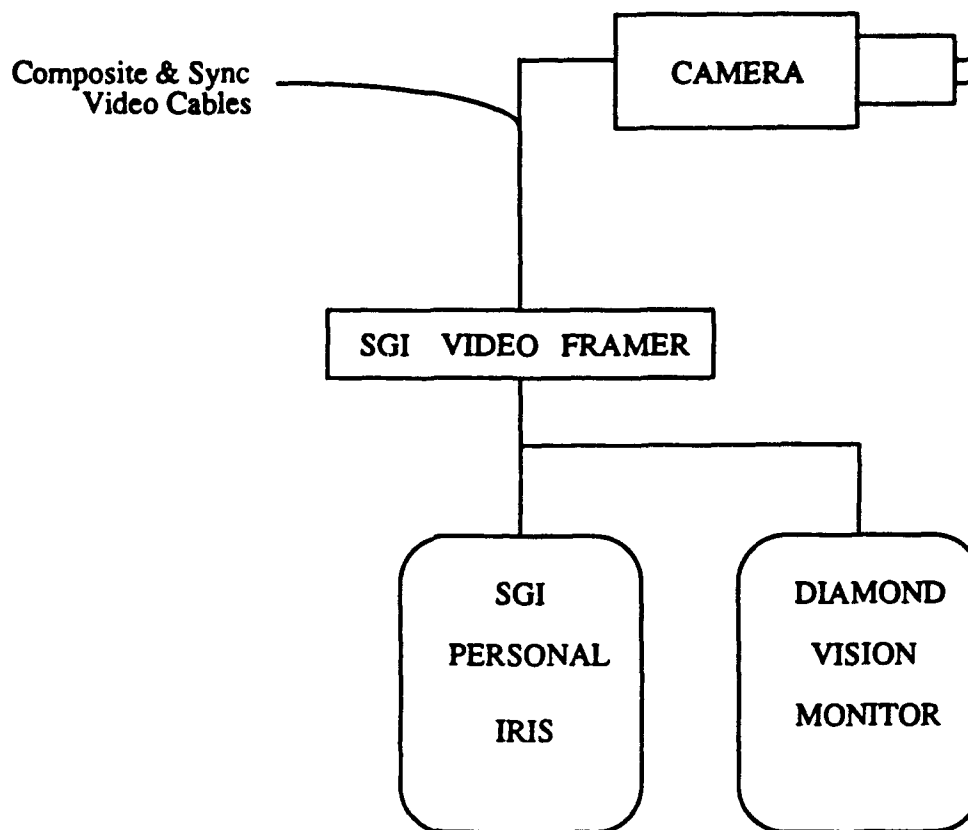
Figure 3.2: Vision system hardware arrangement

operating with the vision system on line, a telephone line is also connected between the processing hardware and the robot, via RS232 ports, eliminating the need for the PowerBook™. This allows interactive simulation of visual interpretation in support of obstacle avoidance to be conducted in lieu of the eventual on-board image processing capability which will be integrated on the VME bus in the future.

B. SOFTWARE

The software system consists of a kernel and a user program. The kernel is approximately 82,000 bytes and only needs to be downloaded once during the course of a given experiment. The user's program can be modified and downloaded quickly to support rapid development.

Motion and sonar commands are issued by the user in MML, the model-based mobile robot language. While the previous version of MML was based on point-to-point tracking, the current version being integrated into *Yamabico*'s control structure relies on a 'path tracking' approach. While MML provides the capability to define path types which include parabolic and cubic spiral, the most fundamental 'path' for the robot to follow is a line which is defined by a curvature (κ) and a location and orientation in two-dimensional space described in x, y, and theta. With κ=0, the line is straight, and κ≠0 produces a circle of radius $1/\kappa$ (κ<0 ≡ clockwise & κ>0 ≡ counter-clockwise). The location and orientation can be the starting point of a semi-infinite line called a forward line (*fline*), the end point of a semi-infinite line called a backward line (*bline*), or a point and direction along and infinite line (*line*). For all path types, once one has been specified and commanded, the robot performs the required calculations and adjusts the curvature of its motion as necessary. Additionally, transitions between successive paths are performed automatically and autonomously.

The functionality inherent in the MML plays a significant role in developing the capability for the robot to avoid obstacles. Consequently, a portion of this research effort was devoted to implementing some of the core functions in the newly developed 'path tracking' approach to motion control. This method allows for dynamic real-time specification of the proposed robot path based on sensory input and is especially well suited to employing the information generated from object recognition. Since the available information will include not only ranges (which is the sole data provided by sonar) but also dimensions, a complete avoidance maneuver can be determined.

10

As mentioned above, the sonar system is also controlled through the MML. Both raw sonar range returns as well as processed 'global' results incorporating least-squares line fitting are available to the user on board the robot. This capability should prove particularly useful in extending the environment in which the vision system can be applied, and its application is addressed in the discussion of the environmental model.

IV. SYSTEM COMPONENTS

A. THREE-DIMENSIONAL WIRE-FRAME MODEL

One of the key components of the obstacle detection routine as implemented on *Yamabico* is the process by which known or expected edges are filtered from the acquired video input by matching the edges extracted from the video image to corresponding edges in a superimposed model image. In order to accomplish this task, a wire-frame model of the robot's operating environment created by Jim Stein in [STE 92] was employed. The following is a general description of the method by which this model is implemented along with a discussion of the modifications which were required for integration into this work. The modified code is provided in Appendix A and includes all files with a 'model' prefix.

1. Operating Environment

The fifth floor of Spanagel Hall at the Naval Postgraduate School is currently the only environment in which testing of *Yamabico* is conducted. Consequently, a three-dimensional model of this area has been created using precise measurements (within a quarter centimeter). It is based on a hierarchical list structure which consists of a world, various polyhedra within the world (including instances of similar configurations), and polygons (made up of three or more vertices described in terms of local two-dimensional coordinate space) which are connected to form each polyhedron. In defining each polygon, a 'z' coordinate value is included to specify its height. Subsequently, corresponding (x,y) vertices from polygons of differing heights may be connected by edges to form walls or other vertical plane structures. It should be noted that since the height of the vertices of a polygon is limited to a single value, this structure is strictly limited to being a horizontal surface.

As a simple example, a rectangular room would be represented as a world containing one polyhedron, namely the room, and two polygons, the floor described by the four (x,y) vertices of the corners with a height of zero, and the ceiling described by the same

(x,y) vertices but with a height equal to that of the actual ceiling. Finally, each set of corresponding vertices are connected by a pointer which represents a vertical line, in this case the corners from floor to ceiling. It should be noted that under this system, anything added to the world must be described in three-dimensional terms since a minimum of three vertices are required to create a valid polyhedron. This restriction would at first appear to prohibit the inclusion of various items in the real environment such as room placards which are essentially two-dimensional. In fact, by describing such an item with a very small thickness (one tenth of an inch perhaps) this limitation is lifted.

A number of aspects to this model directly impact the degree to which the vision system may be employed. One is that the distinction between status as an obstacle or an enclosure is inherent to the polygon description by the order in which vertices are defined. Specification in a clockwise manner denotes an enclosure while a counter-clockwise order is used for obstacles. Another is the capability to specify a pivoting axis (in the z direction only) and the corresponding degree of rotation about that axis for each polyhedron. Thus a door is not limited to strictly an open or shut condition, and its position may be altered accordingly. A final important facet to the model is the ease with which changes can be made to the modeled environment. The implications here are that fairly accurate object recognition data can be used to update the model on a real-time basis, and additionally a completely new environment may be mapped via automated sonar cartography [MAC93] with subsequent generation of a viable model for use by the vision system in pattern matching routines.

The fifth floor model originally consisted of only the main wall structures, floor, ceiling (with lights), doors, and floor molding and did not include various items such as door placards, bulletin boards, etc., all of which significantly impact the edges which are extracted from a video image. Consequently, accurate measurements of all permanent fixtures which would be picked up via video were taken, and the appropriate structures were created and added to the database for the fifth deck passageway. Subsequent to these

13

modifications, the model could be relied upon for use in filtering out all expected edges from the input images.

2. Two-Dimensional Projection

Although having an accurate model of the robot's operating environment is fundamental to the foundation for object recognition, it would be useless without any means by which to project the three-dimensional view onto the two-dimensional image plane for matching purposes. By applying fundamental three-dimensional mapping techniques and incorporating derived parameters of the video hardware, the two-dimensional perspective from any given position (x,y,z,q) can be projected onto a plane which emulates that of the focal plane of the video camera as shown in Figure 4.1. This
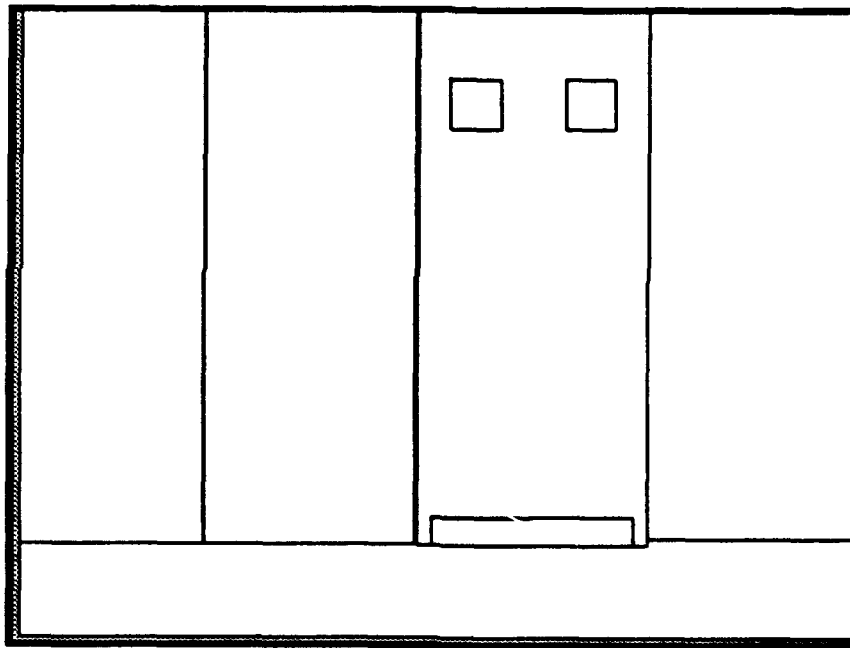
Figure 4.1: Two-dimensional view generated from the three-dimensional wire-frame model

image can then be superimposed upon a video image for visually assessing the match-up, and model line segments can be compared to image line segments on a pixel basis, as is carried out in programs described in Chapter V. To aid in the matching process, model lines

14

are stored in two lists, one for vertical lines and the other for the rest. Each model line is described by both end points (given as x,y pixel coordinates) and pixel length.

3. Coordinate System Transformations

Two coordinate transformations are necessary if location data for an object, generated via the vision system, is to be useful in maneuvering the robot. The first is required because the coordinate system on which the wire-frame model used by the vision system is based differs in both axis direction and origin point from the system now being used in the *Yamabico* project. Figure 4.2 shows the current coordinate system in solid lines
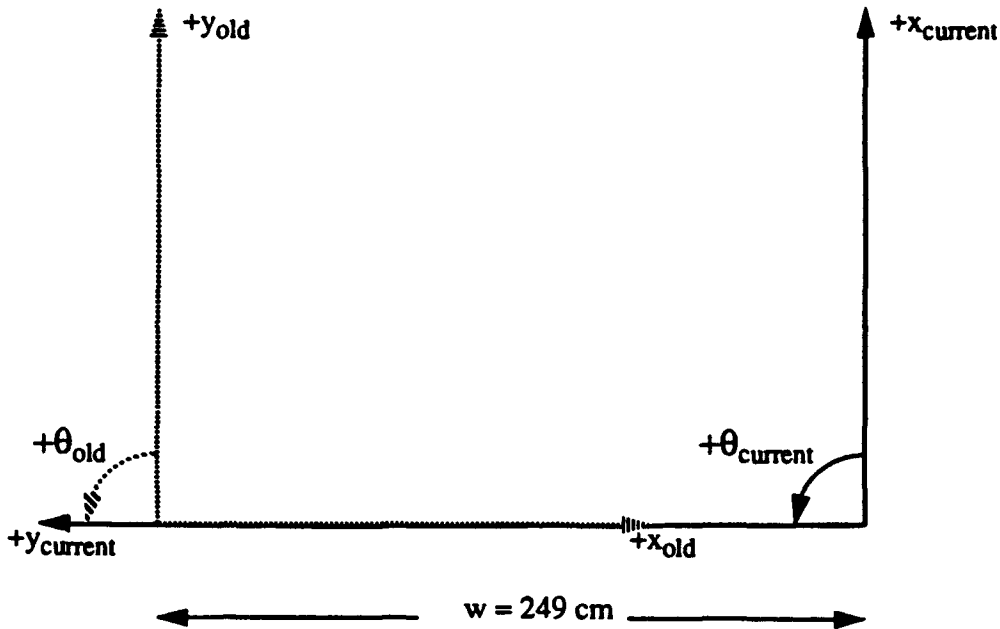
Figure 4.2: Comparison of *Yamabico* operating environment coordinate systems

while the vision system's frame of reference is given in dashed lines. From this drawing it can be seen that for a given (x_1, y_1, θ_1) in the old reference frame, the coordinates in the current system would be $(y_1, w - x_1, \theta_1)$.

15

Once the robot's position is known in the current system, it is desirable to transform vision generated object location data into the global reference frame since all commands for the locomotion control of the robot, as well as all processed sonar data, are given with respect to this coordinate system. Given the robot's position described by a configuration which is comprised of its x, y, and θ values referenced to the current global coordinate system and a local coordinate system fixed on the robot, a location in the robot local system may be described with respect to the global system through the use of the compose function [KAN 93]. Referring to Figure 4.3, if the position of the robot in the
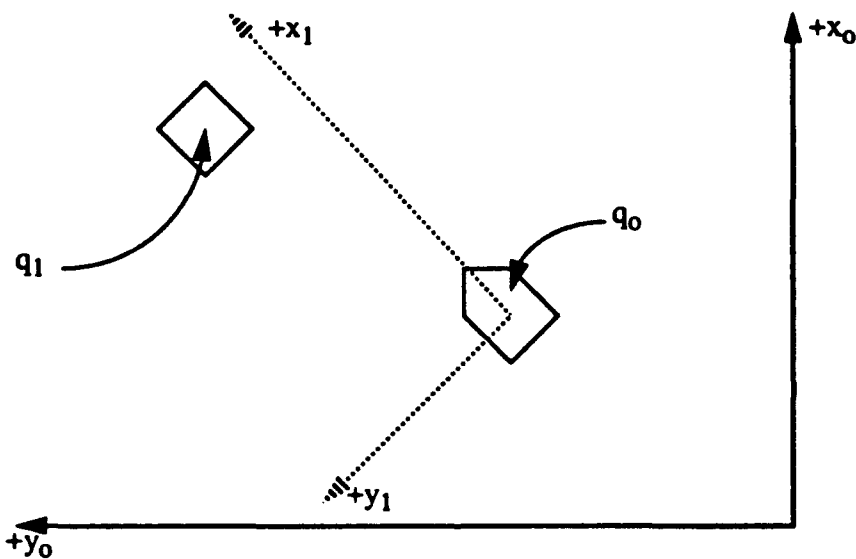
Figure 4.3: Comparison between robot local to environment global coordinate systems

global system is given as q_o and the position of an object in the robot local system as q_1, then the object's position described in the global system will be q and is equal to the composition of q_o with q_1, where the compose function is defined as follows:

16

$$q = q_0 \circ q_1 = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix} \circ \begin{bmatrix} x_1 \\ y_1 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} x_0 + x_1\cos\theta_0 - y_1\sin\theta_0 \\ y_0 + x_1\sin\theta_0 + y_1\cos\theta_0 \\ \theta_0 + \theta_1 \end{bmatrix}$$

Note that in this situation the resulting orientation value $(\theta_0 + \theta_1)$ actually contains no significant information because even though the robot's orientation (θ_0) is usually a known quantity, the orientation of the object (θ_1) is purely arbitrary.

B. EDGE EXTRACTION

The ability to extract edge information which will be matched against the two-dimensional projection generated by the model provides the foundation for being able to analyze the video images seen by the robot. While the initial implementation of vision for *Yamabico* has become obsolete, a portion of the groundwork was applicable to this effort. Namely, the routines coded by Kevin Peterson in [PET92] which provide edge extraction of CCD video camera input (stored as an RGB image) via gradient intensity analysis and application of a least squares method of line determination. As in the case of the wire-frame model, a general description of the implementation is provided and includes those aspects which were modified in order to better support the goals of this work.

1. Pixel Storage and Manipulation

The data generated by the video frame grabber is stored in a 32 bit format with eight bit values for the level of red, green, and blue intensity of each pixel in the image, giving a range of 0 to 255. A conceptual block of storage for describing a single pixel is shown in Figure 4.4. The alpha component, which represents the transparency of the pixel, is not considered when using the RGB format since all pixels are taken to be completely opaque. The data for all the pixels in an image is stored in a long, one-dimensional array which is manipulated by pointers. The ordering in the array with regard to position in the

image is left to right, bottom to top, so the lower left corner pixel would be the first element in the array while the upper right would be the last. Since the edge extraction process, which

ALPHA	BLUE	GREEN	RED

Bits 31   23   15   7   0
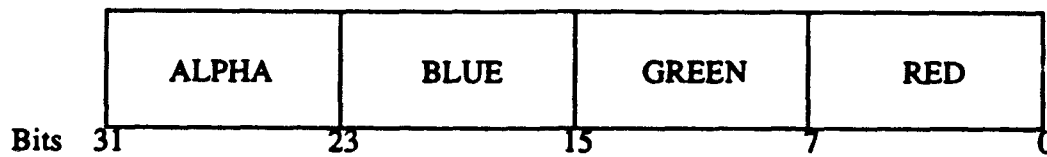
Figure 4.4: 32 Bit pixel storage format

will be described below, requires a black and white ('grayscale') representation for the pixels in an image, a conversion from the RGB values is necessary. According to the standard weighting factors set by the National Television Systems Committee (NTSC), a RGB color pixel is given an equivalent grayscale value by the following relationship:

$$GRAYSCALE = [\ 0.299\ \ 0.587\ \ 0.114\ ] \begin{bmatrix} \text{Red Intensity} \\ \text{Green Intensity} \\ \text{Blue Intensity} \end{bmatrix}$$

An example of a grayscale image is shown in Figure 4.5. It should be noted that since the factor for green dominates over the other two colors, it is possible to reduce the computation involved in this conversion to a single intensity value by basing it on only the green intensity. This, in fact, is how the process was implemented in Peterson's work. After experimentation in the current application, however, it was determined that better image analysis resulted when all three colors were considered. The increased processing time is negligible with respect to the overall analysis and consequently this implementation employs the conversion shown above.

With each pixel now described by a single intensity value, a Sobel operator is applied in order to determine the change in horizontal and vertical intensity with respect to those pixels which surround it. Figure 4.6 shows the Sobel matrix window where the values

18

in the boxes are the factors by which grayscale pixels intensity is multiplied and the boxes themselves represent a pixel with the center box being the pixel to which the operator is

Figure 4.5: Grayscale image of hallway

-1	0	1
$\sqrt{2}$	0	$\sqrt{2}$
-1	0	1

dx

1	$\sqrt{2}$	1
0	0	0
-1	$\sqrt{2}$	-1

dy

Figure 4.6: Sobel matrix window for pixel gradient determination

being applied. A pixel's gradient magnitude and direction is then given by the following relations where the atan2 functions returns the arc tangent in the range $-\pi/2$ to $\pi/2$.

19

$$\text{Gradient Magnitude} = (\,dx^2 + dy^2\,)^{1/2}$$

$$\text{Gradient Direction} = \text{atan2}(\,dy\,,dx\,)$$

2. Edge Determination

The first two of five criterion for line image analysis are now available at this point. In order for a pixel to even be considered for inclusion in an edge region, its gradient magnitude must be above a specified threshold value (C_1). It is possible to construct a 'gradient' image based on this information alone as shown in Figure 4.7. In this format,
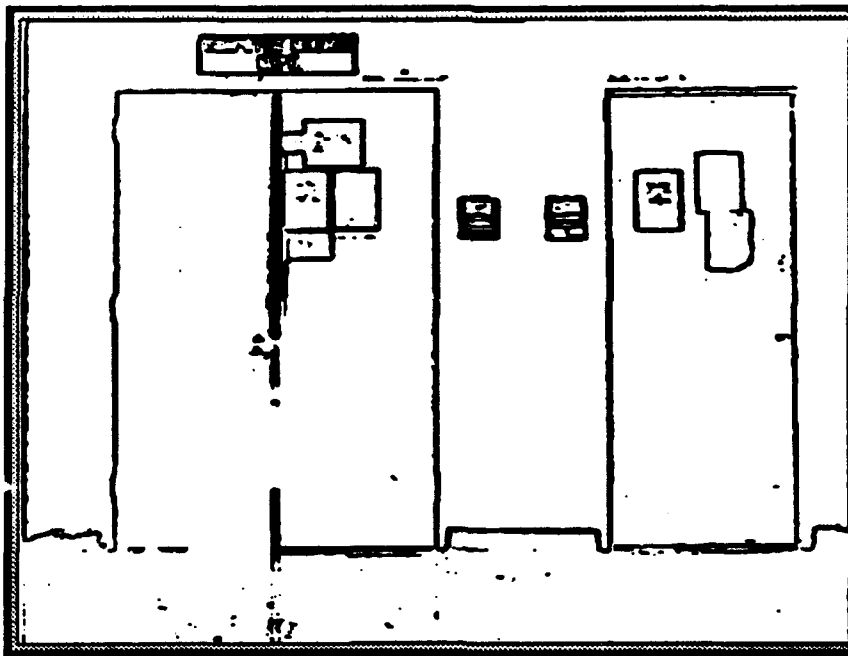
Figure 4.7: Gradient image of hallway

pixels with a gradient magnitude above C_1 are stored as pure black while the rest are stored as pure white. Although this representation is not explicitly used in the subsequent edge generation and line determination, it does provide insight into how well the Sobel is isolating regions of differing light intensity.

20

By grouping together adjacent pixels (which have met the C_1 criteria) with gradient directions which differ by a set angular amount (C_2), edge regions are generated for the entire image. As each pixel is added, the primary moments of inertia (which include the total number of pixels in the edge) are updated for the region. Additionally, once no more pixels are to be added to an edge, it is immediately analyzed for line fitting potential.

3. Least Squares Line Fitting

Once all of the appropriate pixels have been included to form an edge, the region's secondary moments of inertia are computed, making it possible to represent the area as an equivalent ellipse of inertia with a major and minor axis length. Additionally, rho is defined as the ratio between the two axes and describes edge thickness. A line is then fitted by applying the final three criterion which include maximum thickness (C_3), minimum number of pixels (C_4), and minimum length (C_5). Of these, C_3 has the most significant overall impact as a value of 0.1 requires a fairly thin region while a value of 1.0 permits a square blob to be a candidate for line fitting. Figure 4.8 shows the lines which were fitted based on the image in Figure 4.7. After extensive testing, the values used for all the criterion, with the exception of C_1, were modified in order to provide the most useful line segments to the object recognition routines. The code, as modified for this implementation, is provided in Appendix A.

C. VISION SYSTEM

1. Camera Mounting

In mounting the CCD camera on *Yamabico*, a number of variables had to be taken into consideration. Among the less significant concerns was the desire not to interfere with the operation of other robot components and to avoid placement which would significantly alter the confines of the robot structure. Of course, the primary criteria was finding a location which provided the most useful field of view for the task at hand. Due to the hardware limitations of the current camera, the zoom and focus must remain static and

consequently, the depth of image which is generated is constant. This translates into the restriction that the only way to alter the range at which objects initially come into view is
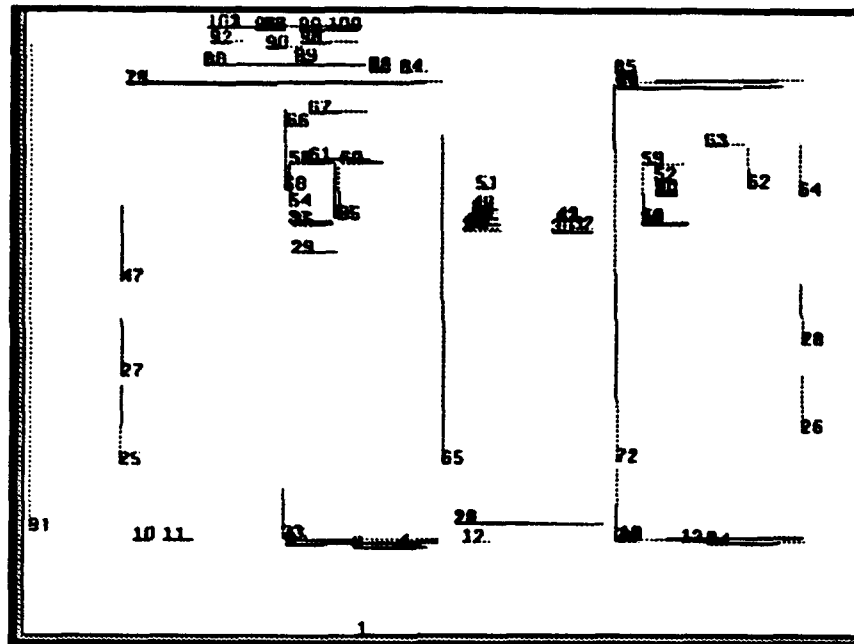
Figure 4.8: Line segments extracted from gradient image in Figure 4.7

by physically altering the line of sight direction of the entire camera. When tilted downward in an effort to pick up objects at fairly close range, two significant problems were introduced. The first resulted from the specular reflection of the florescent lighting system in the passageway upon the tile floor which resulted in a sharp increase in the amount of clutter picked up in the gradient image. Figures 4.9 and 4.10 show the grayscale and gradient images taken from the same location as those taken in Figures 4.5 and 4.7 but with the camera tilted down by approximately 20 degrees. Comparison of Figures 4.7 and 4.10 reveal the significance of this effect. Although the routines which conduct the edge extraction and object detection have some intrinsic filtering effect, this significant increase in the level of clutter would adversely impact subsequent processing.

Secondly, this large tilt angle causes a distortion in the non-vertical, non-horizontal image lines. Since the wire-frame model operates on the premise that the

22
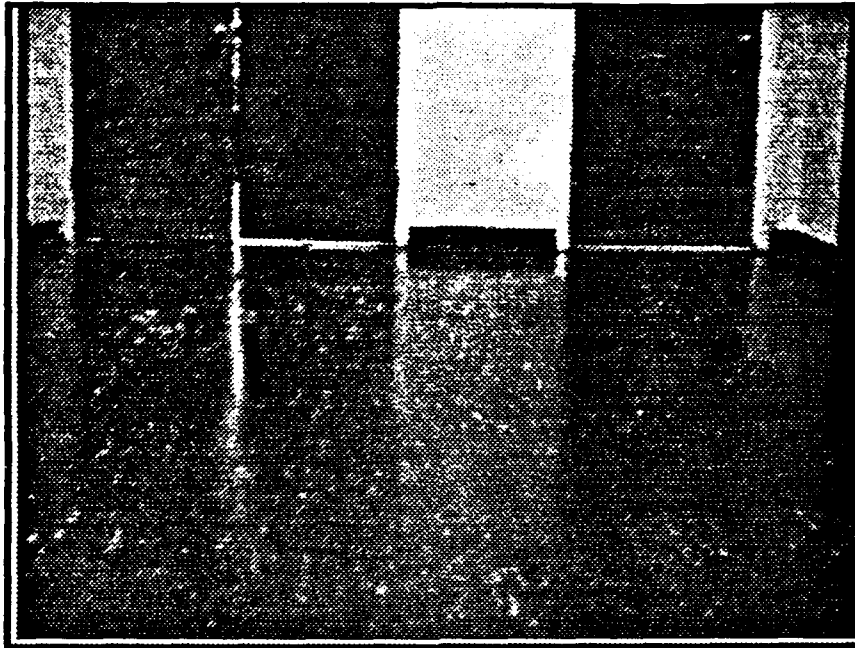
Figure 4.9: Grayscale image of hallway with 20 degree tilt angle
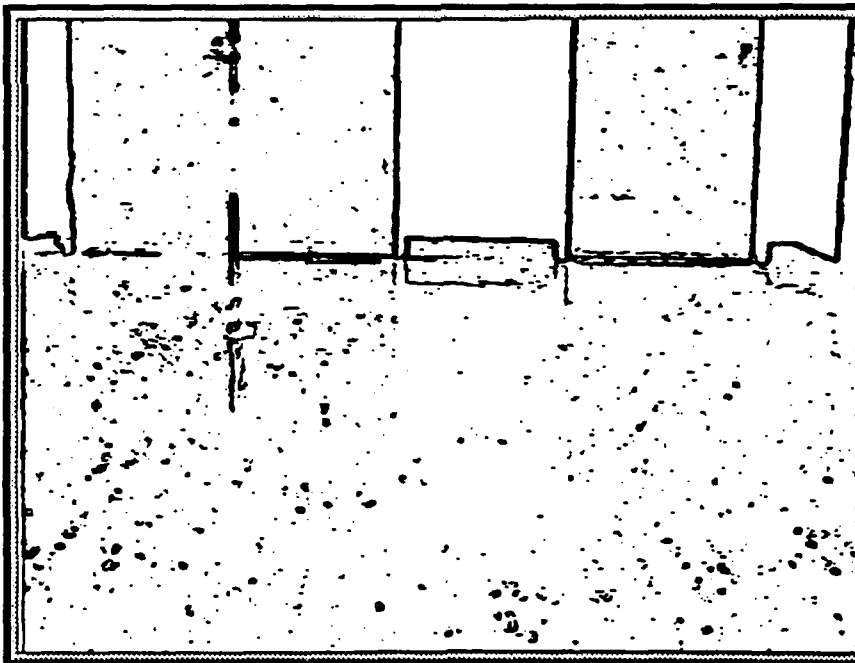
Figure 4.10: Gradient image of hallway with 20 degree tilt angle

viewing is done parallel to the floor, the ability to match lines between the model and image would be seriously impaired.

Obviously, a mounting orientation which is level to the ground, or at least nearly level, is desirable for proper image processing. The concern then turns to what does the camera see and is this useful. After some experimentation, a very satisfactory setup was achieved. With only a slight downward tilt ($< 2°$), no increase in specular reflection is apparent in the images while an object comes completely into view at a range of just under four meters. Since the sonars have a maximum effective range of four meters, this provides a smooth transition between close-in operations to be handled by sonar and long-range planning via vision.

2. Focal Length and Field of View

With the zoom setting held constant, camera focal length will remain static as well. For the camera, as used in this project, the focal length is 4.16 centimeter and its CCD element, or the plane onto which images are projected, is 1.69 centimeters square. Figure 4.11 depicts the physical significance of the focal length (fl), CCD element size, and vertical field of view (FOV) at a range r, if looking at the side of the camera. These factors will be important in performing range and dimension calculations.
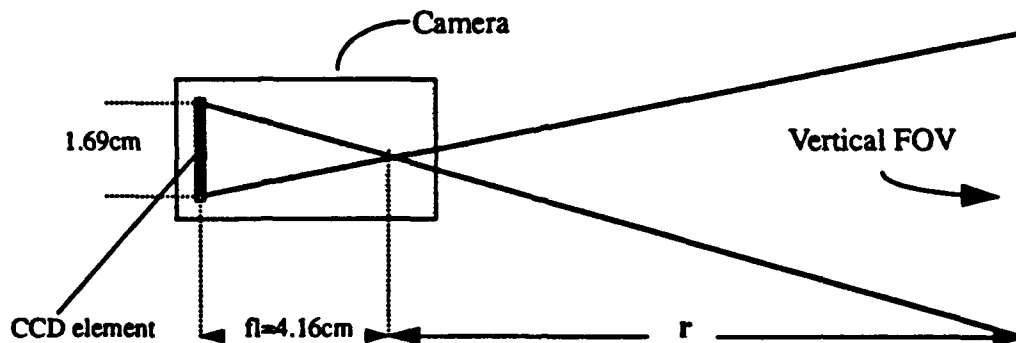
Figure 4.11: Depiction of focal length, CCD element size, and angle delta

V. UNKNOWN OBJECT RECOGNITION

A. EXTRACTING UNKNOWN OBJECTS

With the basic tools described previously, it is now possible to consider the integrated approach for recognizing the presence of an object in any given video image and evaluating the localized object for range and dimension information. Functionally, the entire process can be divided into two parts, image/model processing and object localization. The implementation on *Yamabico* by programs in Appendix B is described in the following sections and includes the details of why a particular action is necessary and how it is carried out.

1. Image/Model Processing

The first thing which must be accomplished is transferring the RGB image data from the video framer storage format into a one-dimensional array structure used during line segment processing. Each element of this array will hold information on a pixel including its red, green, and blue intensity levels, allowing for rapid, sequential analysis of each pixel. A major portion of the image processing, including grayscale conversion, edge determination, and line fitting, may now proceed. The approach, as detailed in Chapter IV, is followed without modification.

As in the case of initial image line generation, the creation of the model view follows the methodology outlined in Chapter IV. A single file is devoted to storing the data describing the fifth floor hallway in which the robot operates and is employed in generating the three-dimensional wire-frame model of this area. The two-dimensional projection of the expected view in this world is then created based on the input robot position and orientation.

Before the information available in the model projection can be applied to the input image for pattern matching, the lines making up the model representation must be filtered. This need arises because the methods inherent in generation of the model list of

25

lines will initially produce lines which are either out of bounds with respect to the image plane area or by their very nature will not be a concern to object detection. This results in wasteful comparisons during matching. The list of two-dimensional model lines is not constrained by the 486 pixel height of the image lines and consequently a model line filtering process is applied to this list which eliminates any lines which lie entirely above a height of 486. Another characteristic of the projection routine which creates the model view is a tendency to generate a number of very short line segments (some that are only a fraction of a pixel in length). Thus, any model line of length less than two pixels is also deleted to reduce the number of comparisons and hence, further cut computation time. One additional test, which is conducted as each model line is analyzed, involves determining the height in the projected view at which the wall meets the floor or the 'horizon'. This simply consists of keeping track of the lowest horizontal line which cuts across the center area of the projection and its usefulness will be covered in the pattern matching discussion.

Additionally, the model lines are not necessarily contiguous segments and in fact what appears as a single line is often a group of adjacent segments. The problem presented by this fragmentation would arise when the actual matching process is initiated because an image line is tested for end point inclusion between the model line's end points. Although a image line may be a valid candidate for filtering, it would be dismissed because it covers a distance greater than each of the individual segments. Therefore, adjacent model lines are combined into one continuous line segment prior to initiation of the matching process.

Facets of the generated image lines as well as their intended application in the object recognition process present the opportunity for elimination prior to the application of the model lines for matching. Unlike the pre-processing conducted on the model lines, all of these checks can be accomplished during the matching of the individual image lines to the list of model lines. The first concerns removal based solely on location in the image plane. Results from a typical line fitting run will include a few lines along the bottom edge of the image frame which carry no significance with regard to any objects. Therefore, image lines which lie entirely below a height of two pixels are marked for deletion. Next,

26

lines which are located along the left and right sides are also set for deletion since the concern of object recognition generally centers on the view directly ahead. The final filter follows concepts outlined in [KAH90] where computation time can and should be reduced by focusing the processing only on the regions of concern. In the case of this work, a very significant area of an image may be disregarded because the robot has a maximum height. Any image lines located entirely above this value will not have an impact on the robot's ability to navigate safely and are therefore marked for deletion.

With all of the model and image line pre-processing completed, the actual elimination of expected image lines by pattern matching with the model lines may be initiated. Each image line is tested against the filter criteria described above, and then it is eliminated either through filtering or by qualifying as a match to a model line with appropriate end point inclusion, or it remains as a valid candidate for being a part of an object's outline. It should be noted that although the model lines are stored in two lists, one for vertical lines and the other for the rest, the image lines are simply contained in a single list in the order in which they were fitted. In pursuit of minimized computation time, it is desirable to only compare a vertical image line to the vertical model lines and likewise for horizontal lines. This is accomplished by determining the image line orientation (vertical, horizontal, or diagonal) through simply assessing its phi value and then performing the matching against the appropriate model list. During this test, the orientation is stored for later use by a sorting routine. For the image shown in Figure 5.1, the initial extracted line segments are presented in Figure 5.2 while Figure 5.3 shows the lines which remain following the pattern matching. Notice that the memos posted on the doors are not eliminated since it would not be appropriate to include them as a permanent part of the hallway model. It should also be noted that although these lines are located well above the maximum height of the robot, they still appear in the final image. This is because any image line (which was not filtered or matched to a model line) located above the robot height will have its respective data structure description annotated to reflect the fact that it is not of concern as an obstacle but is kept in the list of lines for possible post-analysis.
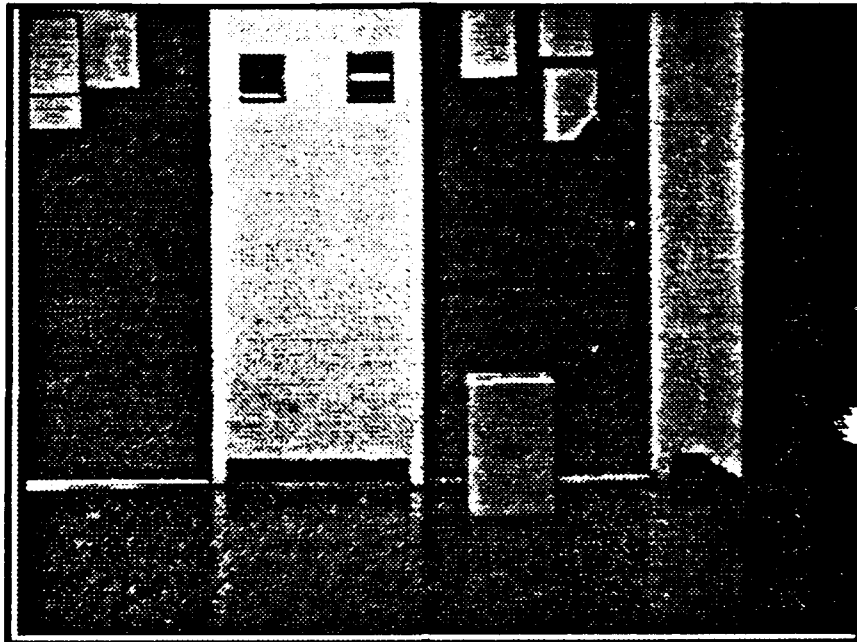
Figure 5.1: Unanalyzed grayscale image
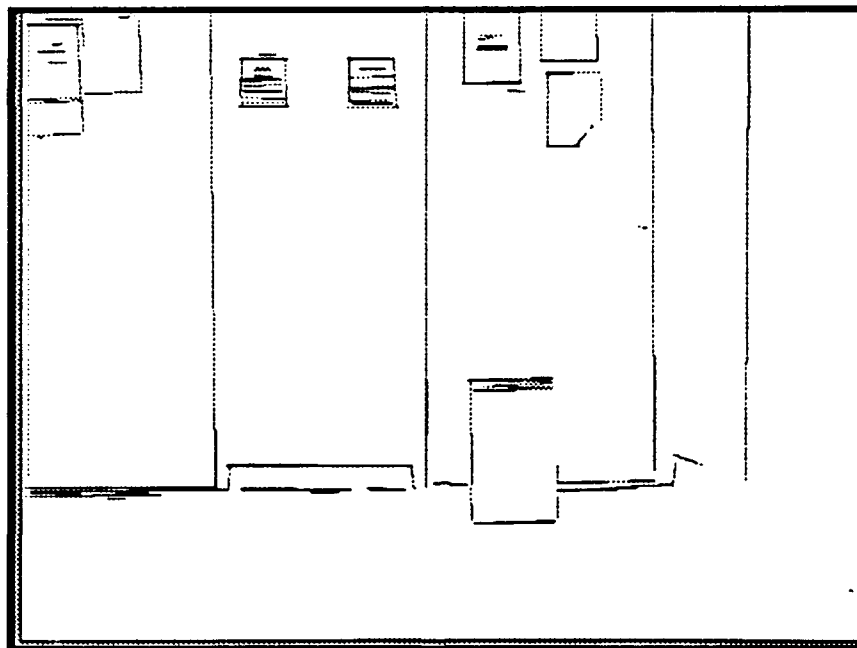
Figure 5.2: Initial line segments for image in Figure 5.1

28
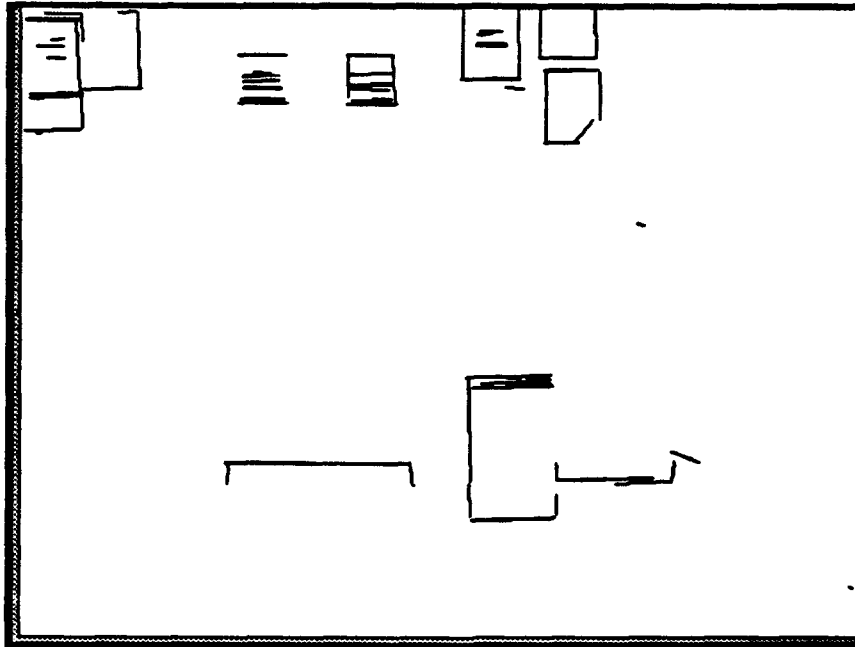
Figure 5.3: Final line segments for image in Figure 5.1 following filtering

An even more ambitious approach to focusing on specific regions may be applied to the elimination process when considering one of the base assumptions which stated that any obstacle encountered will be resting on the floor and not suspended above the floor. It therefore, is a direct consequence that the base of the obstacle will have its edge at a height that in all instances will be lower than the height at which the wall meets the floor. Additionally, the edges which comprise the sides connecting to the base will also have a portion of their length below this level. The height of this meeting point, determined during the model line filtering, is a known value and consequently, it is possible to consider only image lines which exist, at least in part, below this horizon. Figure 5.4 shows the lines which would remain when employing this approach. Even though only a few lines remain, they can provide all the information necessary to effectively carry out object analysis.

2. Object Analysis

Two approaches were pursued in order to properly localize an object in a processed image. The first assumed that the robot would encounter only a single object

29
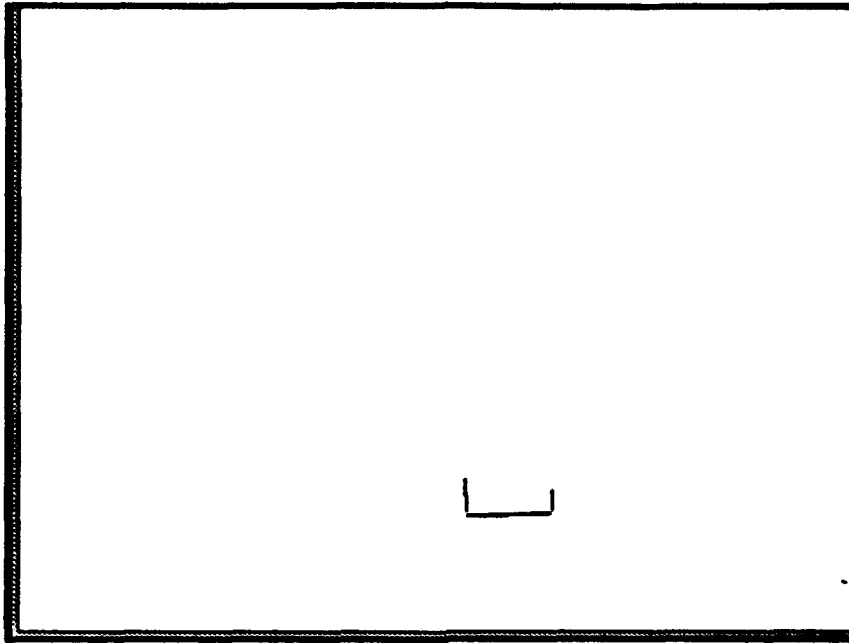
Figure 5.4: Line segments for image in Figure 5.1 after horizon filtering

while the second, which is described in the next section, lifted this restriction. In both cases, the derived information is of the same format and applicable to the subsequent calculations which are necessary for obstacle avoidance. The former method operates on the list of remaining image lines without any further processing. A search is conducted for the left-most vertical line which would be the left edge of the box and a corresponding closest line to the right as the box's right edge. Next, horizontal lines which lie within the confines of the left and right sides are found and designated as the bottom and top of the object. This approach is actually limited in its robustness because the presumption that only one object will be present is compromised by extraneous lines resulting from phenomena such as reflections, shadowing, and light between doors.

3. Multiple Object Filtering

Introducing the possibility that more than one object will be present in an image required both additional processing and a revised approach. Since the number of image lines which are still valid candidates for making up the object outline is relatively small, it

proves worth while to devote the computation time necessary to sort both the vertical and horizontal lines once at the start of object analysis. This eliminates repeated searches through the entire list of lines which would otherwise be required. Following the storage of each line's description in an element of separate arrays for vertical and horizontal lines, an implementation of a quick-sort algorithm as outlined in [MAN91], is applied to position the lines in ascending order, going left to right or bottom to top. This makes it possible to move through the lines only one time in search of the proper combinations.

With the lines sorted, the basic algorithm used to localize those making up an object is as follows: Start with the first vertical line and look at subsequent vertical lines until one is found with the condition that there exists a horizontal line between the two. If no subsequent line was found, move to the next line over and repeat. Once a pair of lines has been found, the element number in the array for the right side line may be saved and searching for the next object continues, beginning with the next element.

B. RANGE DETERMINATION

Yamabico is capable of detecting obstacles at a range of up to four meters using the installed sonar system. However, the robot is essentially 'blind' beyond this range without the benefit of other sensors. Provided that the orthogonal orientation and non-suspension assumptions hold true for each object, it is possible to derive range information from an image generated by the vision system both theoretically and empirically.

1. Theoretical Range Data

Using the physical hardware constants discussed in Chapter IV, it is possible to calculate the range to an object through analysis of the object's base in a video image. Referring to Figure 5.5, β is the angle from the horizontal to the bottom edge of the object, α is the tilt angle of the camera as physically mounted, θ is the angle from the vertical centerline of the camera image (C_L) to the bottom edge of the object, and ρ is the distance from the vertical centerline of the camera image to the bottom edge of the object as

31
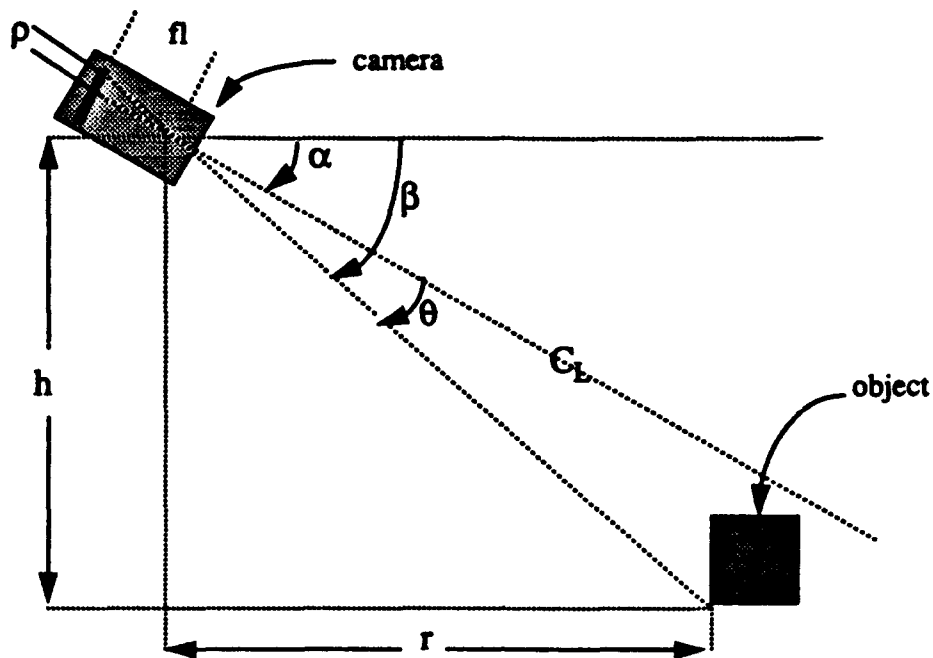
Figure 5.5: Side view depiction of various vision system parameters

measured on the CCD element in pixels. Figure 5.6 depicts how the box would appear in a video image and shows ρ on the image plane. With this information, the range to the object is calculated as follows:

$$\tan(\theta) = \rho/fl \quad \therefore \quad \theta = \arctan(\rho/fl)$$

$$h/r = \tan(\beta) = \tan(\alpha + \theta)$$

Combining and solving for r gives ...

$$r = h / (\tan(\alpha + \arctan(\rho/fl)))$$

A plot of this result, using a tilt angle of 1.83 degrees, is shown in Figure 5.7.

2. Empirical Range Data

Since the base of the object is flush with the floor, the edge at this meeting point will appear in the image as a horizontal line at a given height, measured along the vertical axis. As the range to an object increases, the apparent height of this edge will increase, at a

32
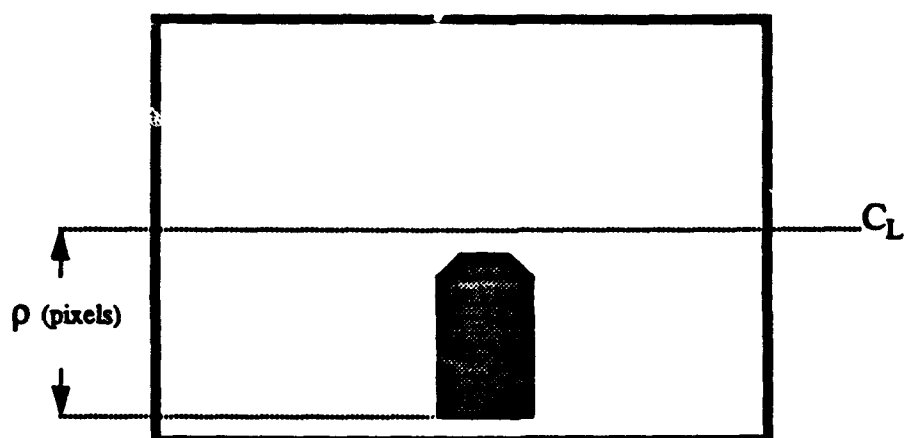
Figure 5.6: Projection of camera view in image plane
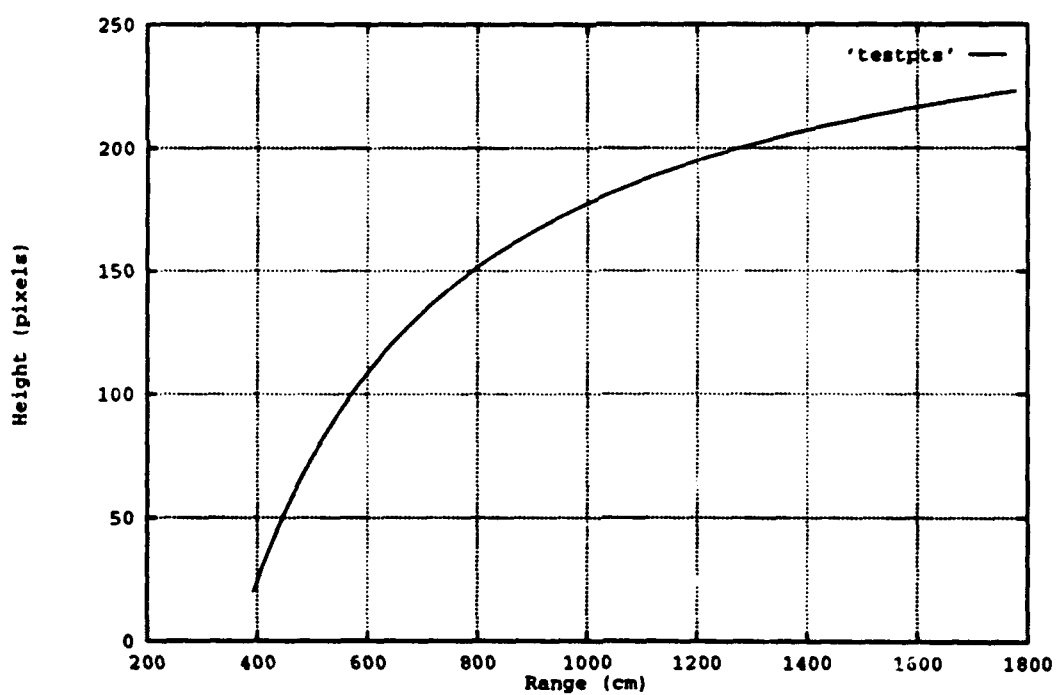
Figure 5.7: Theoretical change in base height with increasing range to object

decreasing rate. Figures 5.8 through 5.11 demonstrate this phenomena, as the first two figures reflect the difference in the position of a box at 4.0 versus 4.5 meters while the second two show the same 0.5 meter change in range, but from 8.0 to 8.5 meters . Clearly

Figure 5.8: Box at a range of 4.0 meters

the change in the vertical height of the box's base is larger at the four meter range than at eight. In fact, a pixel analysis of the images reveals that the base moved from a vertical position of 28.2 to 57.0 pixels at four meters while it only moved from 156.7 to 164.5 pixels at the eight meter range. Figure 5.12 provides a plot of the actual range of a box in centimeters versus the vertical height in pixels of the box's base as seen in an image, much like the theoretical plot of Figure 5.7. Two separate sets of data (taken on the same box but on different days) are shown to demonstrate the fairly consistent nature of this data. As might be expected, the slope of the curve is steeper at shorter distances since a change in the range will be reflected in a noticeable edge height change while at the longer distances a much larger variation in range would be required to detect any significant edge height

.... 57.0

Figure 5.9: Box at a range of 4.5 meters

.... 156.7

Figure 5.10: Box at a range of 8.0 meters

_ 164.5

Figure 5.11: Box at a range of 8.5 meters
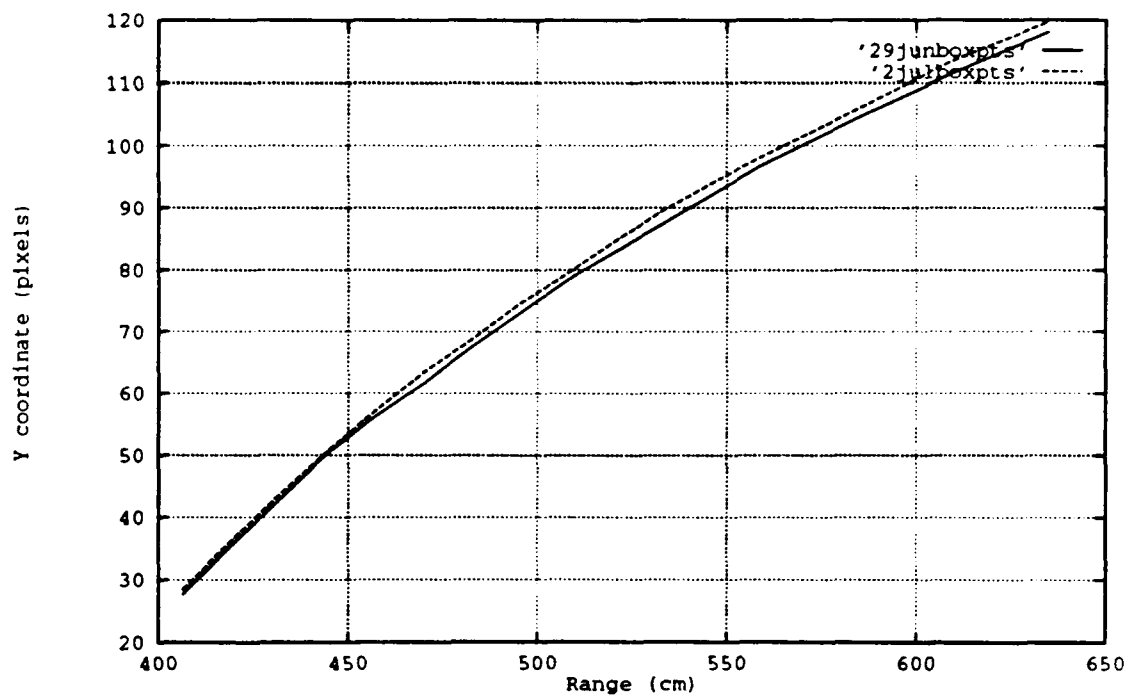
Figure 5.12: Box range versus the height of its base in an image for two separate test runs

36

movement. The tendency of the data runs to drift apart at longer ranges can be attributed to the diminishing accuracy which is caused by this situation.

Although there is no physical cut-off in range to delineate where the accuracies are no longer adequate, it was prudent to make such a determination. A value of ten meters was chosen because variation between differing base heights at this range will introduce no more than a 10 centimeter error, permitting safe navigation of the robot using vision system data. The primary concern, however, centers on how well the theoretical results match up against the raw data. Figure 5.13 provides just such a comparison, with the theoretical
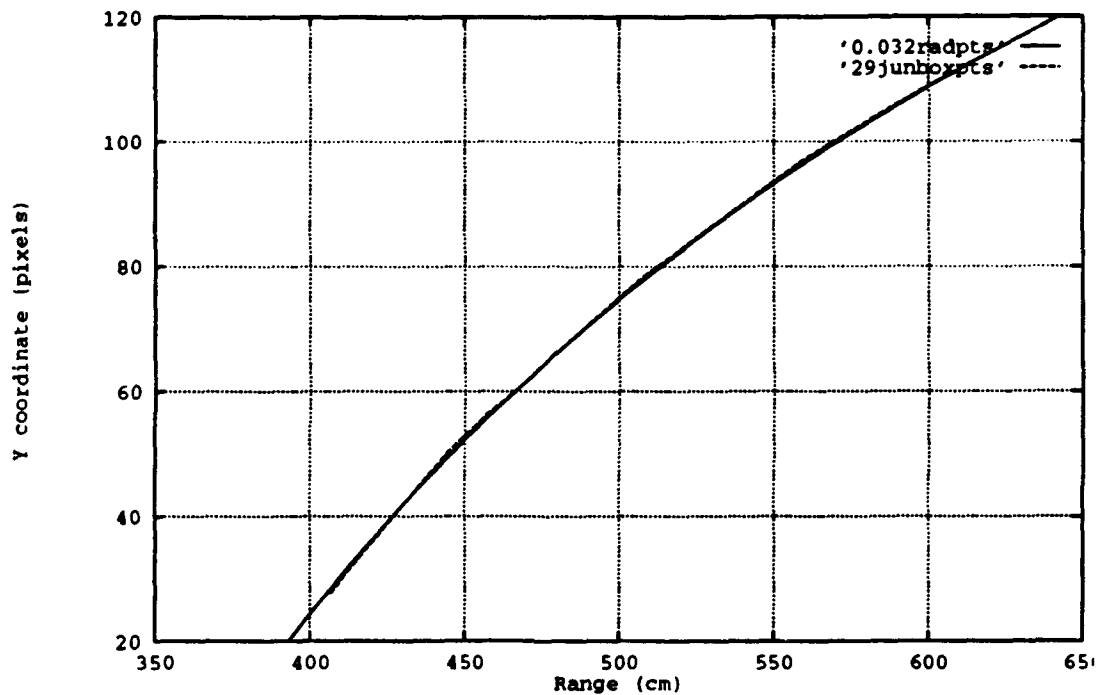
Figure 5.13: Comparison theoretical versus actual data points

points shown plotted as a solid line and actual, image-based points displayed as a dotted line. As can be seen, the derived formula very accurately predicts the actual vision system output, which translates into a precise range determination capability based on visual analysis.

37

C. DIMENSION DETERMINATION

By using the range determined in the previous section and applying trigonometry in a manner similar to its use in the range calculation, it is a relatively simple task to derive dimension information for an object from image analysis. Figure 5.14 depicts an object in
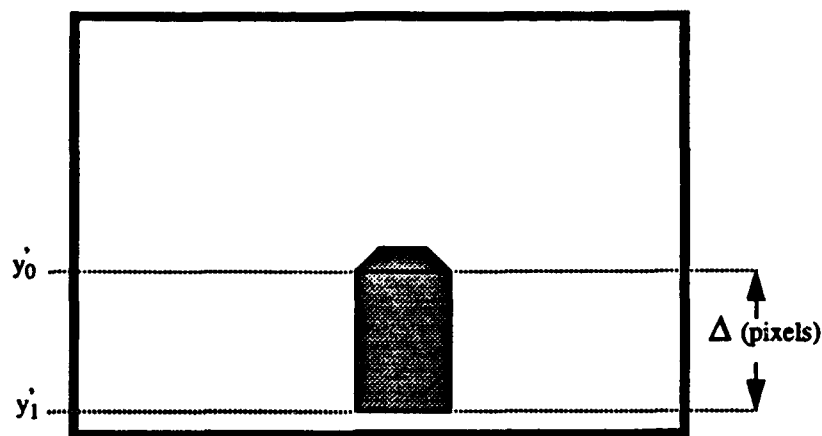
Figure 5.14: View in image plane during determination of height Δ

the image plane with the bottom and top at a vertical position of y_1 and y_0 prime respectively and a distance in-between (or height) of Δ. A side view of this situation is provided in Figure 5.15 and includes the focal length of the camera and the projection of
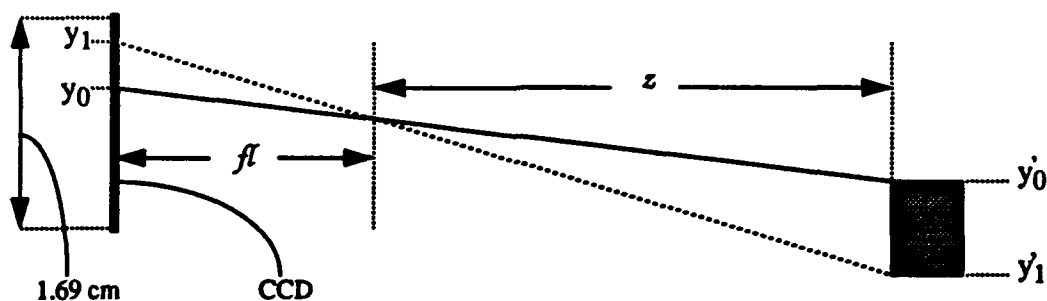
Figure 5.15: Side view of image projection onto CCD element

38

y_0 and y_1 prime as y_0 and y_1 (measured in pixels) from a range z. The following derivation solves for the object height (Δ):

$$\frac{y_0'}{z} = \frac{y_0}{fl} \quad \therefore \quad y_0' = \frac{y_0 \, z}{fl}$$

similarly, $\quad y_1' = \dfrac{y_1 \, z}{fl}$

$$\Delta = y_1' - y_0' = \frac{y_1 \, z}{fl} - \frac{y_0 \, z}{fl} = (y_1 - y_0)\left(\frac{z}{fl}\right)$$

Since y_0 and y_1 are given in pixels, a conversion to centimeters is necessary . . .

CCD physical size $= 1.69$ cm and CCD pixel size $= 486$ pixels

$\therefore \quad$ conversion $= 1.69 / 486 = 0.00348$ cm/pixel

As an example, Figure 5.14 depicts a hallway image with a box placed at a distance of 507 cm from the camera. A pixel analysis reveals that the top and bottom are located along the vertical axis at 48 and 154 pixels while the sides are at 206 and 276 pixels along the horizontal axis. The differences provide a height of 106 pixels and a width of 70 pixels. Applying the above formula, the box dimensions are estimated to be 44x29.5 cm. The actual dimensions of the box are 44x29 cm, so this approach clearly provides very accurate results.
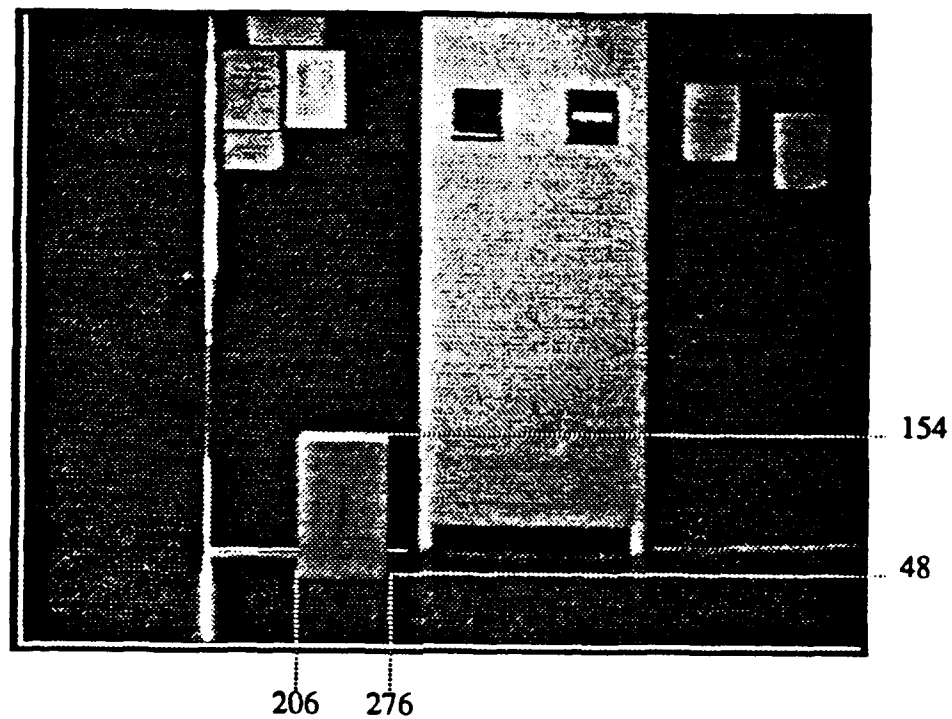
Figure 5.14: Box location in an image described by pixels

VI. OBSTACLE AVOIDANCE

A. IMAGE ANALYSIS INTERPRETATION

Following object evaluation, the left and right sides of the object as known values and the approach to applying this information in obstacle avoidance is fairly simple. Five possible situations can exist with regard to the 'best' path for avoidance by a minimum distance L. The first two cases occur when the entire obstacle is situated more than L to the left or right of the current robot path and consequently, do not require maneuvering. In the third scenario, an obstacle may be positioned such that the current path would bisect it. This necessitates basing the decision to shift left or right on the presence of other obstacles or the proximity to surrounding enclosures. Finally, the obstacle may cross the path, extending more to one side than the other. In these instances, the tendency would be to shift to the side which causes the least significant movement away from the desired trajectory. Once again, other factors might be considered.

In general, however, the objective is still to maneuver around the obstacle by at least L. Since the camera is mounted in line with the center line of the robot, it is reasonable to consider the horizontal center of the images it generates to be in line with the current robot path. The exact same relationship which was used in determining the obstacle's dimensions can also be applied to calculating the distance (D) from the center of the image (or robot path) to the side along which the robot will pass. Corresponding to the cases discussed above and referring to Figure 6.1, a value of $D > L$ with the object entirely to one side would mean that the object will not be of concern while $D < L$ would require a shift by a distance of $L \pm D$. The convention used in the implementation is that a shift to the right is positive and a shift to the left is negative.

B. PATH GENERATION

At this point, the information available from the vision system, as currently implemented, provides the data necessary for the robot to alter its current path such that an

D > L with obstacle entirely to one side -> Obstacle does not force change in path

D < L with obstacle entirely to one side -> Obstacle forces shift of L-D

D < L with obstacle directly in path -> Obstacle forces shift of L+D
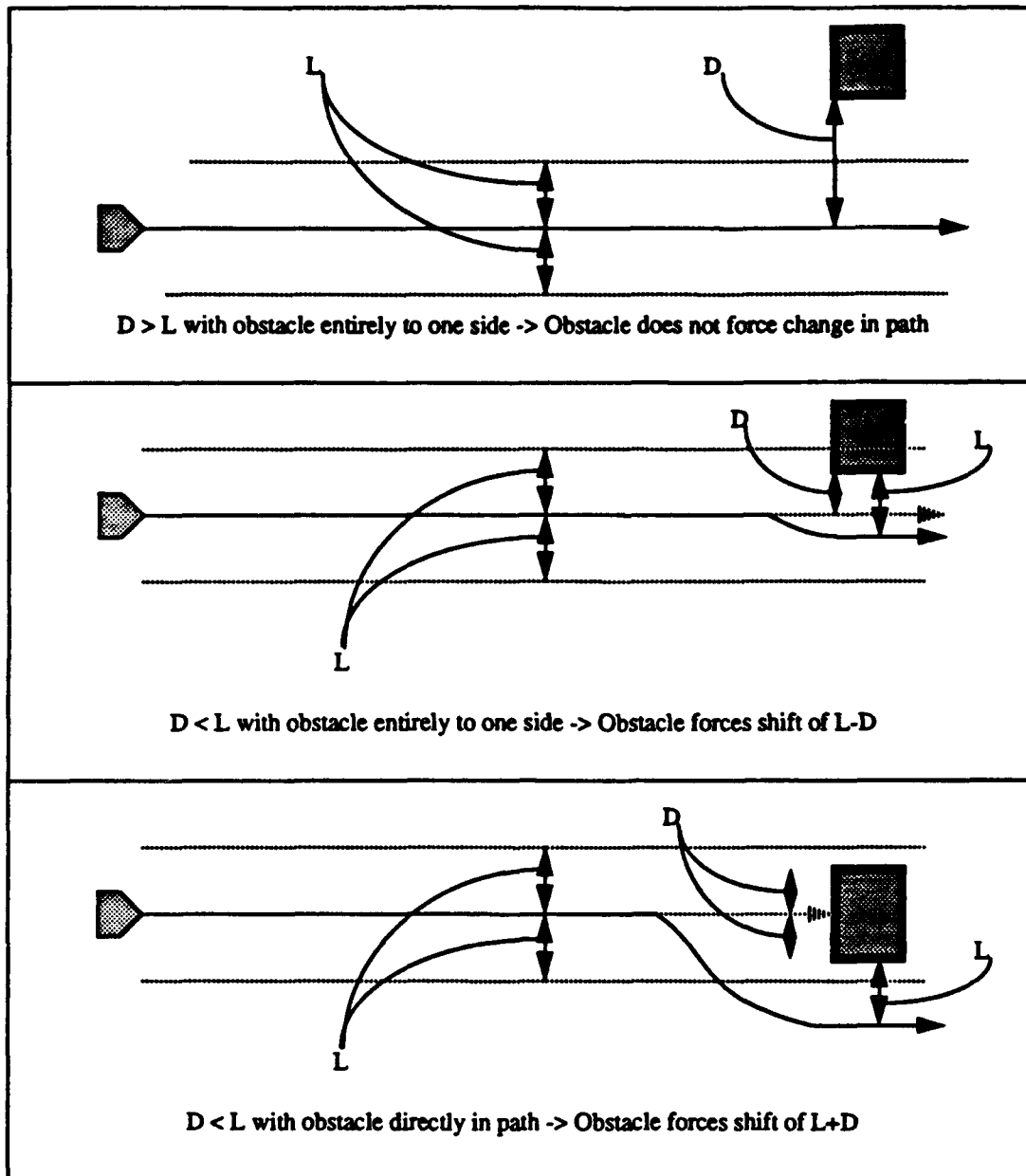
Figure 6.1: Variation in obstacle avoidance maneuvers

obstacle will be avoided by a safe distance L. This essentially amounts to an 'intelligent' lane change. It is possible, however, to enhance this maneuver and return to the original path. By enabling a side looking sonar while passing along side the obstacle, its depth can be determined and a subsequent shift back to the initial path may safely take place. The combination of the two sensors would lead to the following general algorithm:

- Grab an image while traveling along a particular path.

- Perform image analysis on the image.

- If no object is detected, continue on the current path.

- If an object is detected, analyze object for range and dimension information.

- Determine the 'safest' and least significant maneuver and compute the required
  distance to shift left or right.

- Define a new path based on the above input and transition to it.

- Return to the original path once past the object, as detected by side-looking sonar.

Essentially all of the elements in this approach have been covered in detail with the exception of the method used to define a path and the subsequent transition to it. Path definition is a very straight forward task under MML and transitioning between paths is even easier. First, a configuration must be defined. This is accomplished through a call to the function *def_configuration* with input arguments of **x, y, theta, kappa and name**, where the first four arguments are described in Chapter III and **name** is an arbitrary, yet unique, label given to each configuration. Next, a path which will include this configuration is declared via the *line* function with a single input argument of the configuration name. Once *line* has been invoked, MML will automatically determine a smooth path for transitioning to the new path as well as provide the locomotion control necessary for steering the robot along these paths.

As an example, if avoidance requires moving left or right a distance Δ to a parallel path as shown in Figure 6.2, the actual commands to the robot would be as follows, where

43

a call to the *get_rob0* function will provide the current robot configuration in the global coordinate system:

def_configuration(0, −Δ, 0, 0, &local)

get_rob0(&config)

line(compose(&config, &local, &config))

There are two important aspects to note in this series of commands. First, the call to *def_configuration* simply defines an orientation which is physically Δ to the right or left of the robot. This is based on the local frame of reference where x is in the direction of motion and corresponds to a theta of $0°$, leaving the y axis to describe the distance to either side. Second, as discussed in Chapter IV, the compose function must be employed in order to convert this locally based orientation derived from the image analysis into the global frame of reference in which the robot operates. The shift back to the original path would require the identical three calls with the exception of using Δ in the call to *def_configuration*.
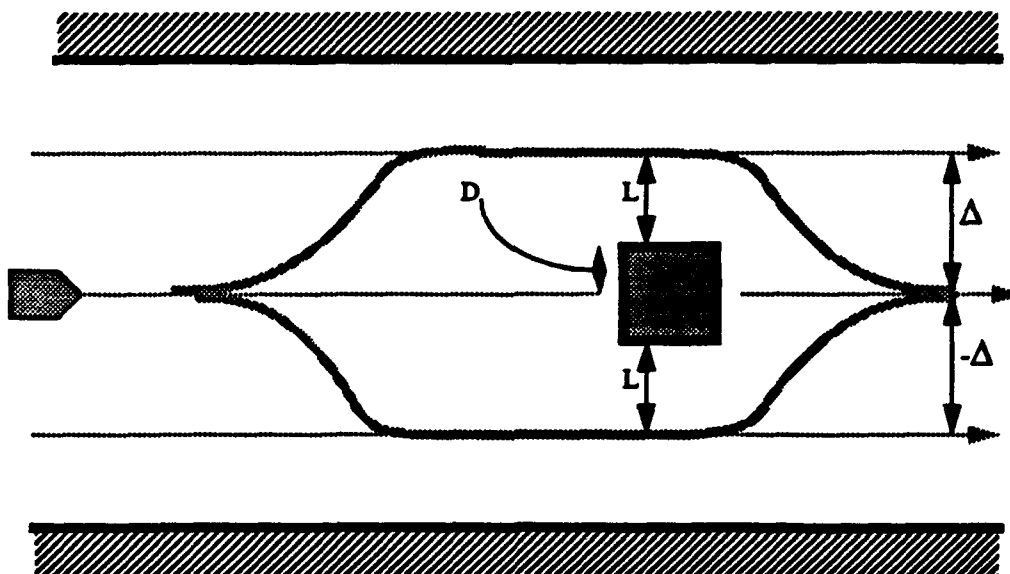
Figure 6.2: Obstacle avoidance employing image analysis input for situation described in Figure 6.1.c

44

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

1. Results

Experiments employing the integrated system were successful in a number of regards. The accuracies with respect to dimension and range determination certainly exceed the expectations which were present upon commencing this work and the pattern matching implementation appears to provide consistent object recognition. Unfortunately, testing under fully autonomous conditions was not feasible due to the lack of an on-board image processing capability. Even without the capability to identify obstacles and ascertain depth information, the robot gains increased knowledge about the world in which it is operating compared to operating solely off of sonar output.

2. Concerns

Despite the generally favorable results from this implementation, a number of factors are still a concern. As with most vision systems, variations in lighting can have an adverse effect on the ability to properly extract line segments, resulting in improper object analysis. Another aspect which may hinder the processing is the dependency on accurate position information for generating the expected view from the model. Robot odometry is subject to errors due to floor unevenness and wheel slippage which accumulate with time if left uncorrected. The need for stable robot locomotion to prevent fluctuations in the tilt angle is also a critical factor, although *Yamabico's* movements are very smooth. Finally, if the robot is travelling at medium to high speed, the processing time may prohibit the image understanding system from providing the necessary avoidance information rapidly enough to ensure a safe and timely maneuver.

B. RECOMMENDATIONS

1. Hardware

Obviously, the biggest physical limitation to this system is the requirement to have video cables attached to the robot in order to grab an image. Although the ultimate goal is to perform all processing on board the robot with dedicated hardware specifically designed for this task, a wireless video link would be an economical, yet practical interim solution. An added benefit to this upgrade is that when the processing capability is available on the robot, the wireless link could still be used for remotely monitoring what the robot is seeing. A pivoting camera, possibly with auto-focus capability, would certainly enhance the usefulness of vision as a sensor by drastically reducing the field of view and the directional limitations imposed by a static mounting and the constant focus. Additionally, an inclinometer mounted to the vehicle could be used to provide accurate tilt angle measurements.

2. Software

Another potential approach to verifying the tilt angle would be to base its calculation on interpretation of each image. Just as range to an object is computed by the height of its base in an image, the reverse could be done, in that knowledge of the height where the wall meets the floor and the range to the wall is sufficient for determining the tilt angle. Probably the most promising concept is the use of the recently implemented automated cartography capability for generating a three-dimensional wire-frame model of a previously unknown environment. By mapping the walls which enclose the robot and transforming this data into a 'global' frame of reference, the model building routines could be directly invoked to create a three-dimensional world model with an arbitrary value for the height of the enclosure, thus permitting use of the image understanding system in what was previously an unknown environment.

APPENDIX A - MODEL AND EDGE EXTRACTION ROUTINES

The following routines provide implementation for the creation of the fifth floor model, the transformation of a three-dimensional view from a position in the model onto a two-dimensional plane, image storage structures, and edge extraction of video images. The files included are the following:
model5th.h, modelgraphics.h, modelvisibility.h, model2d+d.h, edgesupport.h, npsimagesuport.h

```
/* FILE:            5th.h

   AUTHORS:         LT James Stein / LT Mark DeClue

   THESIS ADVISOR: Dr. Kanayama

   CALLS TO FILES: 2d+.h


   COMMENTS: This is the construction file for the 2d+ model of the
5th floor Spanagel Hall (1st half only - up to glass double doors). All
coordinates are in inches while all angles are in degrees.
                 The main function "make_world" is called to
build the model using function calls to file 2d+.h. Type definitions for
WORLD, POLYHEDRON, POLYGON, and VERTEX can be found at the top of this file
also.
   Notice that the floor of H1 is one huge, concave polygon which
makes up the floor to the hallway as well as all of the office floors. To
this
floor numerous ceilings are added for offices, door jams, and main
corridors.
Doors, lights, and molding strips are then added to the model as separate
polyhedra.
*/

WORLD *make_world()
{
  WORLD *W;
  POLYHEDRON *H1, *H2, *H3, *H4, *H5, *H6, *H7, *H8, *H9, *H10, *H11, *H12,
    *H13, *H14, *H15, *H16, *H17, *H18, *H19, *H20, *H21, *H22, *H23, *H24,
    *H25, *H26, *H27, *H28, *H29, *H30, *H31, *H32, *H33, *H34, *H35, *H36,
    *H37, *H38, *H39, *H40, *H41, *H42, *H43, *H44, *H45, *H46, *H47, *H48;
  POLYGON *H1P1, *H1P2, *H1P3,
    *H1P4, *H1P5 ,*H1P6, *H1P7, *H1P8, *H1P9, *H1P10, *H1P11, *H1P12,
    *H1P13, *H1P14, *H1P15, *H1P16, *H1P17, *H1P18, *H1P19, *H1P20, *H1P21,
    *H1P22, *H1P23, *H1P24, *H1P25, *H1P26, *H1P27, *H1P28, *H1P29, *H1P30,
    *H1P31, *H1P32, *H1P33, *H1P34, *H1P35, *H1P36, *H1P37, *H1P38, *H1P39,
    *H1P40, *H1P41, *H1P42, *H1P43, *H1P44, *H1P45, *H1P46, *H1P47, *H1P48,
```

```
*H1P49, *H1P50, *H1P51, *H1P52, *H1P53, *H1P54, *H1P55, *H1P56, *H1P57,
 *H1P58, *H1P59, *H1P60, *H1P61, *H1P62, *H1P63, *H1P64, *H1P65,

 *H2P1, *H2P2, *H3P1, *H3P2, *H4P1, *H4P2, *H5P1, *H5P2,
 *H6P1, *H7P1, *H7P2, *H8P1, *H8P2, *H9P1, *H9P2, *H10P1, *H10P2,
*H11P1, *H11P2, *H12P1, *H12P2, *H13P1, *H13P2, *H14P1, *H14P2, *H15P1,
*H15P2, *H16P1, *H16P2, *H17P1, *H17P2, *H18P1, *H18P2, *H19P1, *H19P2,
*H20P1, *H20P2,
*H21P1, *H21P2, *H22P1, *H22P2, *H23P1, *H23P2, *H24P1, *H24P2, *H25P1,
*H25P2, *H26P1, *H26P2, *H27P1, *H27P2, *H28P1, *H28P2, *H29P1, *H29P2,
*H30P1, *H30P2,
*H31P1, *H31P2, *H32P1, *H32P2, *H33P1, *H33P2, *H34P1, *H34P2, *H35P1,
*H35P2, *H36P1, *H36P2, *H37P1, *H37P2, *H38P1, *H38P2, *H39P1, *H39P2,
*H40P1, *H40P2, *H41P1, *H41P2, *H42P1, *H42P2, *H43P1, *H43P2, *H44P1,
*H44P2, *H45P1, *H45P2, *H46P1, *H46P2, *H47P1, *H47P2, *H48P1, *H48P2,
last_p;

VERTEX *H1P1V1, *H1P1V2, *H1P1V3, *H1P1V4, *H1P1V5, *H1P1V6, *H1P1V7,
     *H1P1V8,*H1P1V9,*H1P1V10, *H1P1V11, *H1P1V12, *H1P1V13, *H1P1V14,
        *H1P1V15,*H1P1V16,*H1P1V17, *H1P1V18, *H1P1V19, *H1P1V20,
*H1P1V21,
        *H1P1V22,*H1P1V23,*H1P1V24, *H1P1V25, *H1P1V26, *H1P1V27,
*H1P1V28,
        *H1P1V29,*H1P1V30,*H1P1V31, *H1P1V32, *H1P1V33, *H1P1V34,
*H1P1V35,
        *H1P1V36,*H1P1V37,*H1P1V38, *H1P1V39, *H1P1V40, *H1P1V41,
*H1P1V42,
        *H1P1V43,*H1P1V44,*H1P1V45, *H1P1V46, *H1P1V47, *H1P1V48,
*H1P1V49,
        *H1P1V50,*H1P1V51,*H1P1V52, *H1P1V53, *H1P1V54, *H1P1V55,
*H1P1V56,
        *H1P1V57,*H1P1V58,*H1P1V59, *H1P1V60, *H1P1V61, *H1P1V62,
*H1P1V63,
        *H1P1V64,*H1P1V65,*H1P1V66, *H1P1V67, *H1P1V68, *H1P1V69,
*H1P1V70,

        *H1P1V2a, *H1P1V2b, *H1P1V2c, *H1P1V2d, *H1P1V2e, *H1P1V2f,
        *H1P1V4a, *H1P1V4b, *H1P1V4c, *H1P1V4d, *H1P1V4e, *H1P1V4f,
        *H1P1V6a, *H1P1V6b, *H1P1V6c, *H1P1V6d, *H1P1V6e, *H1P1V6f,
        *H1P1V8a, *H1P1V8b, *H1P1V8c, *H1P1V8d, *H1P1V8e, *H1P1V8f,
       *H1P1V10a, *H1P1V10b, *H1P1V10c, *H1P1V10d, *H1P1V10e, *H1P1V10f,
       *H1P1V12a, *H1P1V12b, *H1P1V12c, *H1P1V12d, *H1P1V12e, *H1P1V12f,
       *H1P1V14a, *H1P1V14b, *H1P1V14c, *H1P1V14d, *H1P1V14e, *H1P1V14f,
       *H1P1V16a, *H1P1V16b, *H1P1V16c, *H1P1V16d, *H1P1V16e, *H1P1V16f,
       *H1P1V18a, *H1P1V18b, *H1P1V18c, *H1P1V18d, *H1P1V18e, *H1P1V18f,
       *H1P1V20a, *H1P1V20b, *H1P1V20c, *H1P1V20d, *H1P1V20e, *H1P1V20f,
        *H1P1V22a, *H1P1V22b,
       *H1P1V24a, *H1P1V24b, *H1P1V24c, *H1P1V24d, *H1P1V24e, *H1P1V24f,
       *H1P1V26a, *H1P1V26b, *H1P1V26c, *H1P1V26d, *H1P1V26e, *H1P1V26f,
       *H1P1V28a, *H1P1V28b, *H1P1V28c, *H1P1V28d, *H1P1V28e, *H1P1V28f,
       *H1P1V30a, *H1P1V30b, *H1P1V30c, *H1P1V30d, *H1P1V30e, *H1P1V30f,
```

*H1P1V32a, *H1P1V32b, *H1P1V32c, *H1P1V32d, *H1P1V32e, *H1P1V32f,
*H1P1V34a, *H1P1V34b, *H1P1V34c, *H1P1V34d, *H1P1V34e, *H1P1V34f,
*H1P1V36a, *H1P1V36b, *H1P1V36c, *H1P1V36d, *H1P1V36e, *H1P1V36f,
*H1P1V38a, *H1P1V38b, *H1P1V38c, *H1P1V38d, *H1P1V38e, *H1P1V38f,
*H1P1V40a, *H1P1V40b, *H1P1V40c, *H1P1V40d, *H1P1V40e, *H1P1V40f,
*H1P1V42a, *H1P1V42b, *H1P1V42c, *H1P1V42d, *H1P1V42e, *H1P1V42f,
*H1P1V44a, *H1P1V44b, *H1P1V44c, *H1P1V44d, *H1P1V44e, *H1P1V44f,
*H1P1V46a, *H1P1V46b, *H1P1V46c, *H1P1V46d, *H1P1V46e, *H1P1V46f,
*H1P1V48a, *H1P1V48b, *H1P1V48c, *H1P1V48d, *H1P1V48e, *H1P1V48f,
*H1P1V50a, *H1P1V50b, *H1P1V50c, *H1P1V50d, *H1P1V50e, *H1P1V50f,
*H1P1V52a, *H1P1V52b, *H1P1V52c, *H1P1V52d, *H1P1V52e, *H1P1V52f,
*H1P1V55a, *H1P1V55b, *H1P1V55c, *H1P1V55d, *H1P1V55e, *H1P1V55f,
*H1P1V58a, *H1P1V58b, *H1P1V58c, *H1P1V58d, *H1P1V58e, *H1P1V58f,
*H1P1V60a, *H1P1V60b, *H1P1V60c, *H1P1V60d, *H1P1V60e, *H1P1V60f,
*H1P1V63a, *H1P1V63b, *H1P1V63c, *H1P1V63d, *H1P1V63e, *H1P1V63f,
 *H1P1V63g,
*H1P1V65a, *H1P1V65b, *H1P1V65c, *H1P1V65d, *H1P1V65e, *H1P1V65f,
 *H1P1V65g,
*H1P1V68a, *H1P1V68b, *H1P1V68c, *H1P1V68d, *H1P1V68e, *H1P1V68f,

 *H1P2V1, *H1P2V2, *H1P2V3, *H1P2V4,
    *H1P3V1, *H1P3V2, *H1P3V3, *H1P3V4,
 *H1P4V1, *H1P4V2, *H1P4V3, *H1P4V4,
 *H1P5V1, *H1P5V2, *H1P5V3, *H1P5V4,
 *H1P6V1, *H1P6V2, *H1P6V3, *H1P6V4,
 *H1P7V1, *H1P7V2, *H1P7V3, *H1P7V4,
 *H1P8V1, *H1P8V2, *H1P8V3, *H1P8V4,
 *H1P9V1, *H1P9V2, *H1P9V3, *H1P9V4,
 *H1P10V1, *H1P10V2, *H1P10V3, *H1P10V4,
 *H1P11V1, *H1P11V2, *H1P11V3, *H1P11V4,
 *H1P12V1, *H1P12V2, *H1P12V3, *H1P12V4,
 *H1P13V1, *H1P13V2, *H1P13V3, *H1P13V4,
 *H1P14V1, *H1P14V2, *H1P14V3, *H1P14V4,
 *H1P15V1, *H1P15V2, *H1P15V3, *H1P15V4,
 *H1P16V1, *H1P16V2, *H1P16V3, *H1P16V4,
 *H1P17V1, *H1P17V2, *H1P17V3, *H1P17V4,
 *H1P18V1, *H1P18V2, *H1P18V3, *H1P18V4,
 *H1P19V1, *H1P19V2, *H1P19V3, *H1P19V4,
 *H1P20V1, *H1P20V2, *H1P20V3, *H1P20V4,
 *H1P21V1, *H1P21V2, *H1P21V3, *H1P21V4,
 *H1P22V1, *H1P22V2, *H1P22V3, *H1P22V4,
 *H1P23V1, *H1P23V2, *H1P23V3, *H1P23V4,
 *H1P24V1, *H1P24V2, *H1P24V3, *H1P24V4,
 *H1P25V1, *H1P25V2, *H1P25V3, *H1P25V4,
 *H1P26V1, *H1P26V2, *H1P26V3, *H1P26V4,
 *H1P27V1, *H1P27V2, *H1P27V3, *H1P27V4,
 *H1P28V1, *H1P28V2, *H1P28V3, *H1P28V4,
 *H1P29V1, *H1P29V2, *H1P29V3, *H1P29V4,
 *H1P30V1, *H1P30V2, *H1P30V3, *H1P30V4,
 *H1P31V1, *H1P31V2, *H1P31V3, *H1P31V4, *H1P31V5,
 *H1P32V1, *H1P32V2, *H1P32V3, *H1P32V4, *H1P32V5,

*H1P33V1, *H1P33V2, *H1P33V3, *H1P33V4,
*H1P34V1, *H1P34V2, *H1P34V3, *H1P34V4,
*H1P35V1, *H1P35V2, *H1P35V3, *H1P35V4,
*H1P36V1, *H1P36V2, *H1P36V3, *H1P36V4,
*H1P37V1, *H1P37V2, *H1P37V3, *H1P37V4,
*H1P38V1, *H1P38V2, *H1P38V3, *H1P38V4,
*H1P39V1, *H1P39V2, *H1P39V3, *H1P39V4,
*H1P40V1, *H1P40V2, *H1P40V3, *H1P40V4,
*H1P41V1, *H1P41V2, *H1P41V3, *H1P41V4,
*H1P42V1, *H1P42V2, *H1P42V3, *H1P42V4,
*H1P43V1, *H1P43V2, *H1P43V3, *H1P43V4,
*H1P44V1, *H1P44V2, *H1P44V3, *H1P44V4,
*H1P45V1, *H1P45V2, *H1P45V3, *H1P45V4,
*H1P46V1, *H1P46V2, *H1P46V3, *H1P46V4,
*H1P47V1, *H1P47V2, *H1P47V3, *H1P47V4,
*H1P48V1, *H1P48V2, *H1P48V3, *H1P48V4,
*H1P49V1, *H1P49V2, *H1P49V3, *H1P49V4,
*H1P50V1, *H1P50V2, *H1P50V3, *H1P50V4,
*H1P51V1, *H1P51V2, *H1P51V3, *H1P51V4,
*H1P52V1, *H1P52V2, *H1P52V3, *H1P52V4,
*H1P53V1, *H1P53V2, *H1P53V3, *H1P53V4,
*H1P54V1, *H1P54V2, *H1P54V3, *H1P54V4,
*H1P55V1, *H1P55V2, *H1P55V3, *H1P55V4,
*H1P56V1, *H1P56V2, *H1P56V3, *H1P56V4,
*H1P57V1, *H1P57V2, *H1P57V3, *H1P57V4,
*H1P58V1, *H1P58V2, *H1P58V3, *H1P58V4,
*H1P59V1, *H1P59V2, *H1P59V3, *H1P59V4,
*H1P60V1, *H1P60V2, *H1P60V3, *H1P60V4,
*H1P61V1, *H1P61V2, *H1P61V3, *H1P61V4,
*H1P62V1, *H1P62V2, *H1P62V3, *H1P62V4,
*H1P63V1, *H1P63V2, *H1P63V3, *H1P63V4,
*H1P64V1, *H1P64V2, *H1P64V3, *H1P64V4,
*H1P65V1, *H1P65V2, *H1P65V3, *H1P65V4,

*H2P1V1, *H2P1V2, *H2P1V3, *H2P1V4, *H2P2V1, *H2P2V2, *H2P2V3, *H2P2V4,
*H3P1V1, *H3P1V2, *H3P1V3, *H3P1V4, *H3P2V1, *H3P2V2, *H3P2V3, *H3P2V4,
*H4P1V1, *H4P1V2, *H4P1V3, *H4P1V4, *H4P2V1, *H4P2V2, *H4P2V3, *H4P2V4,
*H5P1V1, *H5P1V2, *H5P1V3, *H5P1V4, *H5P2V1, *H5P2V2, *H5P2V3, *H5P2V4,
*H6P1V1, *H6P1V2, *H6P1V3, *H6P1V4,
*H7P1V1, *H7P1V2, *H7P1V3, *H7P1V4, *H7P2V1, *H7P2V2, *H7P2V3, *H7P2V4,
*H8P1V1, *H8P1V2, *H8P1V3, *H8P1V4, *H8P2V1, *H8P2V2, *H8P2V3, *H8P2V4,
*H9P1V1, *H9P1V2, *H9P1V3, *H9P1V4, *H9P2V1, *H9P2V2, *H9P2V3, *H9P2V4,
*H10P1V1, *H10P1V2, *H10P1V3, *H10P1V4, *H10P2V1, *H10P2V2, *H10P2V3,
*H10P2V4,
*H11P1V1, *H11P1V2, *H11P1V3, *H11P1V4, *H11P2V1, *H11P2V2, *H11P2V3,
*H11P2V4,
*H12P1V1, *H12P1V2, *H12P1V3, *H12P1V4, *H12P2V1, *H12P2V2, *H12P2V3,
*H12P2V4,
*H13P1V1, *H13P1V2, *H13P1V3, *H13P1V4, *H13P2V1, *H13P2V2, *H13P2V3,
*H13P2V4,
*H14P1V1, *H14P1V2, *H14P1V3, *H14P1V4, *H14P2V1, *H14P2V2, *H14P2V3,

```
*H14P2V4,
*H15P1V1, *H15P1V2, *H15P1V3, *H15P1V4, *H15P2V1, *H15P2V2, *H15P2V3,
*H15P2V4,
*H16P1V1, *H16P1V2, *H16P1V3, *H16P1V4, *H16P2V1, *H16P2V2, *H16P2V3,
*H16P2V4,
*H17P1V1, *H17P1V2, *H17P1V3, *H17P1V4, *H17P2V1, *H17P2V2, *H17P2V3,
*H17P2V4,
*H18P1V1, *H18P1V2, *H18P1V3, *H18P1V4, *H18P2V1, *H18P2V2, *H18P2V3,
*H18P2V4,
*H19P1V1, *H19P1V2, *H19P1V3, *H19P1V4, *H19P2V1, *H19P2V2, *H19P2V3,
*H19P2V4,
*H20P1V1, *H20P1V2, *H20P1V3, *H20P1V4, *H20P2V1, *H20P2V2, *H20P2V3,
*H20P2V4,
*H21P1V1, *H21P1V2, *H21P1V3, *H21P1V4, *H21P2V1, *H21P2V2, *H21P2V3,
*H21P2V4,
*H22P1V1, *H22P1V2, *H22P1V3, *H22P1V4, *H22P2V1, *H22P2V2, *H22P2V3,
*H22P2V4,
*H23P1V1, *H23P1V2, *H23P1V3, *H23P1V4, *H23P2V1, *H23P2V2, *H23P2V3,
*H23P2V4,
*H24P1V1, *H24P1V2, *H24P1V3, *H24P1V4, *H24P2V1, *H24P2V2, *H24P2V3,
*H24P2V4,
*H25P1V1, *H25P1V2, *H25P1V3, *H25P1V4, *H25P2V1, *H25P2V2, *H25P2V3,
*H25P2V4,
*H26P1V1, *H26P1V2, *H26P1V3, *H26P1V4, *H26P2V1, *H26P2V2, *H26P2V3,
*H26P2V4,
*H27P1V1, *H27P1V2, *H27P1V3, *H27P1V4, *H27P2V1, *H27P2V2, *H27P2V3,
*H27P2V4,
*H28P1V1, *H28P1V2, *H28P1V3, *H28P1V4, *H28P2V1, *H28P2V2, *H28P2V3,
*H28P2V4,
*H29P1V1, *H29P1V2, *H29P1V3, *H29P1V4, *H29P2V1, *H29P2V2, *H29P2V3,
*H29P2V4,
*H30P1V1, *H30P1V2, *H30P1V3, *H30P1V4, *H30P2V1, *H30P2V2, *H30P2V3,
*H30P2V4,
*H31P1V1, *H31P1V2, *H31P1V3, *H31P1V4, *H31P2V1, *H31P2V2, *H31P2V3,
*H31P2V4,
*H32P1V1, *H32P1V2, *H32P1V3, *H32P1V4, *H32P2V1, *H32P2V2, *H32P2V3,
*H32P2V4,
*H33P1V1, *H33P1V2, *H33P1V3, *H33P1V4, *H33P2V1, *H33P2V2, *H33P2V3,
*H33P2V4,
*H34P1V1, *H34P1V2, *H34P1V3, *H34P1V4, *H34P2V1, *H34P2V2, *H34P2V3,
*H34P2V4,
*H35P1V1, *H35P1V2, *H35P1V3, *H35P1V4, *H35P2V1, *H35P2V2, *H35P2V3,
*H35P2V4,
*H36P1V1, *H36P1V2, *H36P1V3, *H36P1V4, *H36P2V1, *H36P2V2, *H36P2V3,
*H36P2V4,
*H37P1V1, *H37P1V2, *H37P1V3, *H37P1V4, *H37P2V1, *H37P2V2, *H37P2V3,
*H37P2V4,
*H38P1V1, *H38P1V2, *H38P1V3, *H38P1V4, *H38P2V1, *H38P2V2, *H38P2V3,
*H38P2V4,
*H39P1V1, *H39P1V2, *H39P1V3, *H39P1V4, *H39P2V1, *H39P2V2, *H39P2V3,
*H39P2V4,
```

```
*H40P1V1,  *H40P1V2,  *H40P1V3,  *H40P1V4,  *H40P2V1,  *H40P2V2,  *H40P2V3,
*H40P2V4,
*H41P1V1,  *H41P1V2,  *H41P1V3,  *H41P1V4,  *H41P2V1,  *H41P2V2,  *H41P2V3,
*H41P2V4,
*H42P1V1,  *H42P1V2,  *H42P1V3,  *H42P1V4,  *H42P2V1,  *H42P2V2,  *H42P2V3,
*H42P2V4,
*H43P1V1,  *H43P1V2,  *H43P1V3,  *H43P1V4,  *H43P2V1,  *H43P2V2,  *H43P2V3,
*H43P2V4,
*H44P1V1,  *H44P1V2,  *H44P1V3,  *H44P1V4,  *H44P2V1,  *H44P2V2,  *H44P2V3,
*H44P2V4,
*H45P1V1,  *H45P1V2,  *H45P1V3,  *H45P1V4,  *H45P2V1,  *H45P2V2,  *H45P2V3,
*H45P2V4,
*H46P1V1,  *H46P1V2,  *H46P1V3,  *H46P1V4,  *H46P2V1,  *H46P2V2,  *H46P2V3,
*H46P2V4,
*H47P1V1,  *H47P1V2,  *H47P1V3,  *H47P1V4,  *H47P2V1,  *H47P2V2,  *H47P2V3,
*H47P2V4,
*H48P1V1,  *H48P1V2,  *H48P1V3,  *H48P1V4,  *H48P2V1,  *H48P2V2,  *H48P2V3,
*H48P2V4,
last_v;

  W=add_world("5th_floor",9);
  H1=add_ph("front_hall",10,W,1,0);
  H1P1=add_pg(H1,0.0,1,0);
  H1P1V1 = add_vertex(H1P1,0.0,0.0);
  H1P1V2 = add_vertex(H1P1,0.0,239.5);   /*rm 506*/
  H1P1V2a = add_vertex(H1P1,-5.3,239.5);
  H1P1V2b = add_vertex(H1P1,-5.3,203.3);
  H1P1V2c = add_vertex(H1P1,-244.1,203.3);
  H1P1V2d = add_vertex(H1P1,-244.1,309.4);
  H1P1V2e = add_vertex(H1P1,-5.3,309.4);
  H1P1V2f = add_vertex(H1P1,-5.3,275.2);

  H1P1V3 = add_vertex(H1P1,0.0,275.2);
  H1P1V4 = add_vertex(H1P1,0.0,713.7);   /*rm 510*/
  H1P1V4a = add_vertex(H1P1,-5.3,713.7);
  H1P1V4b = add_vertex(H1P1,-5.3,677.5);
  H1P1V4c = add_vertex(H1P1,-244.1,677.5);
  H1P1V4d = add_vertex(H1P1,-244.1,783.6);
  H1P1V4e = add_vertex(H1P1,-5.3,783.6);
  H1P1V4f = add_vertex(H1P1,-5.3,749.4);

  H1P1V5 = add_vertex(H1P1,0.0,749.4);
  H1P1V6 = add_vertex(H1P1,0.0,825.9);   /* rm 512*/
  H1P1V6a = add_vertex(H1P1,-5.3,825.9);
  H1P1V6b = add_vertex(H1P1,-5.3,789.7);
  H1P1V6c = add_vertex(H1P1,-244.1,789.7);
  H1P1V6d = add_vertex(H1P1,-244.1,895.8);
  H1P1V6e = add_vertex(H1P1,-5.3,895.8);
  H1P1V6f = add_vertex(H1P1,-5.3,861.6);

  H1P1V7 = add_vertex(H1P1,0.0,861.6);
```

```
H1P1V8 = add_vertex(H1P1,0.0,937.5);   /* rm 514*/
H1P1V8a = add_vertex(H1P1,-5.3,937.5);
H1P1V8b = add_vertex(H1P1,-5.3,901.3);
H1P1V8c = add_vertex(H1P1,-244.1,901.3);
H1P1V8d = add_vertex(H1P1,-244.1,1007.4);
H1P1V8e = add_vertex(H1P1,-5.3,1007.4);
H1P1V8f = add_vertex(H1P1,-5.3,973.2);

H1P1V9 = add_vertex(H1P1,0.0,973.2);
H1P1V10 = add_vertex(H1P1,0.0,1049.7);   /* rm 516 */
H1P1V10a = add_vertex(H1P1,-5.3,1049.7);
H1P1V10b = add_vertex(H1P1,-5.3,1013.5);
H1P1V10c = add_vertex(H1P1,-244.1,1013.5);
H1P1V10d = add_vertex(H1P1,-244.1,1119.6);
H1P1V10e = add_vertex(H1P1,-5.3,1119.6);
H1P1V10f = add_vertex(H1P1,-5.3,1085.4);

H1P1V11 = add_vertex(H1P1,0.0,1085.4);
H1P1V12= add_vertex(H1P1,0.0,1161.7);   /* rm 518 */
H1P1V12a = add_vertex(H1P1,-5.3,1161.7);
H1P1V12b = add_vertex(H1P1,-5.3,1125.5);
H1P1V12c = add_vertex(H1P1,-244.1,1125.5);
H1P1V12d = add_vertex(H1P1,-244.1,1231.6);
H1P1V12e = add_vertex(H1P1,-5.3,1231.6);
H1P1V12f = add_vertex(H1P1,-5.3,1197.4);

H1P1V13 = add_vertex(H1P1,0.0,1197.4);
H1P1V14 = add_vertex(H1P1,0.0,1273.4);   /* rm 520 */
H1P1V14a = add_vertex(H1P1,-5.3,1273.4);
H1P1V14b = add_vertex(H1P1,-5.3,1237.2);
H1P1V14c = add_vertex(H1P1,-244.1,1237.2);
H1P1V14d = add_vertex(H1P1,-244.1,1343.3);
H1P1V14e = add_vertex(H1P1,-5.3,1343.3);
H1P1V14f = add_vertex(H1P1,-5.3,1309.1);

H1P1V15 = add_vertex(H1P1,0.0,1309.1);
H1P1V16 = add_vertex(H1P1,0.0,1429.6);   /* rm 522R */
H1P1V16a = add_vertex(H1P1,-5.3,1429.6);
H1P1V16b = add_vertex(H1P1,-5.3,1393.4);
H1P1V16c = add_vertex(H1P1,-244.1,1393.4);
H1P1V16d = add_vertex(H1P1,-244.1,1499.5);
H1P1V16e = add_vertex(H1P1,-5.3,1499.5);
H1P1V16f = add_vertex(H1P1,-5.3,1461.3);

H1P1V17 = add_vertex(H1P1,0.0,1461.3);
H1P1V18 = add_vertex(H1P1,0.0,1488.0);   /* FD #1 */
H1P1V18a = add_vertex(H1P1,-5.5,1488.0);
H1P1V18b = add_vertex(H1P1,-5.5,1486.0);
H1P1V18c = add_vertex(H1P1,-50.0,1486.0);
H1P1V18d = add_vertex(H1P1,-50.0,1562.0);
H1P1V18e = add_vertex(H1P1,-5.5,1562.0);
```

```
H1P1V18f = add_vertex(H1P1,-5.5,1560.0);

H1P1V19 = add_vertex(H1P1,0.0,1560.0);
H1P1V20 = add_vertex(H1P1,0.0,1583.3);     /* rm 524 */
H1P1V20a = add_vertex(H1P1,-5.3,1583.3);
H1P1V20b = add_vertex(H1P1,-5.3,1547.1);
H1P1V20c = add_vertex(H1P1,-244.1,1547.1);
H1P1V20d = add_vertex(H1P1,-244.1,1653.2);
H1P1V20e = add_vertex(H1P1,-5.3,1653.2);
H1P1V20f = add_vertex(H1P1,-5.3,1619.0);

H1P1V21 = add_vertex(H1P1,0.0,1619.0);
H1P1V22 = add_vertex(H1P1,0.0,1650.4);     /* water cooler */
H1P1V22a = add_vertex(H1P1,-30.0,1650.4);
H1P1V22b = add_vertex(H1P1,-30.0,1684.5);
H1P1V23 = add_vertex(H1P1,0.0,1684.5);
H1P1V24 = add_vertex(H1P1,0.0,1754.5);     /* rm 526R */
H1P1V24a = add_vertex(H1P1,-5.3,1754.5);
H1P1V24b = add_vertex(H1P1,-5.3,1718.3);
H1P1V24c = add_vertex(H1P1,-244.1,1718.3);
H1P1V24d = add_vertex(H1P1,-244.1,1790.0);
H1P1V24e = add_vertex(H1P1,-5.3,1790.0);
H1P1V24f = add_vertex(H1P1,-5.3,1786.2);

H1P1V25 = add_vertex(H1P1,0.0,1786.2);
H1P1V26 = add_vertex(H1P1,0.0,1836.4);     /* rm 528A */
H1P1V26a = add_vertex(H1P1,-5.3,1836.4);
H1P1V26b = add_vertex(H1P1,-5.3,1800.2);
H1P1V26c = add_vertex(H1P1,-244.1,1800.2);
H1P1V26d = add_vertex(H1P1,-244.1,1875.0);
H1P1V26e = add_vertex(H1P1,-5.3,1875.0);
H1P1V26f = add_vertex(H1P1,-5.3,1872.1);

H1P1V27 = add_vertex(H1P1,0.0,1872.1);
H1P1V28 = add_vertex(H1P1,0.0,1919.1);     /* rm 528B */
H1P1V28a = add_vertex(H1P1,-5.3,1919.1);
H1P1V28b = add_vertex(H1P1,-5.3,1882.9);
H1P1V28c = add_vertex(H1P1,-244.1,1882.9);
H1P1V28d = add_vertex(H1P1,-244.1,1989.0);
H1P1V28e = add_vertex(H1P1,-5.3,1989.0);
H1P1V28f = add_vertex(H1P1,-5.3,1954.8);

H1P1V29 = add_vertex(H1P1,0.0,1954.8);
H1P1V30 = add_vertex(H1P1,0.0,2030.4);     /* rm 530A */
H1P1V30a = add_vertex(H1P1,-5.3,2030.4);
H1P1V30b = add_vertex(H1P1,-5.3,1994.2);
H1P1V30c = add_vertex(H1P1,-244.1,1994.2);
H1P1V30d = add_vertex(H1P1,-244.1,2100.3);
H1P1V30e = add_vertex(H1P1,-5.3,2100.3);
H1P1V30f = add_vertex(H1P1,-5.3,2066.1);
```

```
H1P1V31 = add_vertex(H1P1,0.0,2066.1);
H1P1V32 = add_vertex(H1P1,0.0,2195.1);   /* rm 530B */
H1P1V32a = add_vertex(H1P1,-5.3,2195.1);
H1P1V32b = add_vertex(H1P1,-5.3,2158.8);
H1P1V32c = add_vertex(H1P1,-244.1,2158.8);
H1P1V32d = add_vertex(H1P1,-244.1,2250.0);
H1P1V32e = add_vertex(H1P1,-5.3,2250.0);
H1P1V32f = add_vertex(H1P1,-5.3,2230.8);

H1P1V33 = add_vertex(H1P1,0.0,2230.8);
H1P1V34 = add_vertex(H1P1,0.0,2253.8);   /* rm 530C */
H1P1V34a = add_vertex(H1P1,-5.3,2253.8);
H1P1V34b = add_vertex(H1P1,-5.3,2251.0);
H1P1V34c = add_vertex(H1P1,-244.1,2251.0);
H1P1V34d = add_vertex(H1P1,-244.1,2350.0);
H1P1V34e = add_vertex(H1P1,-5.3,2350.0);
H1P1V34f = add_vertex(H1P1,-5.3,2289.5);

H1P1V35 = add_vertex(H1P1,0.0,2289.5);
H1P1V36 = add_vertex(H1P1,0.0,2351.2);
H1P1V37 = add_vertex(H1P1,98.0,2351.2);
H1P1V38 = add_vertex(H1P1,98.0,2171.9);   /* rm 421 */
H1P1V38a = add_vertex(H1P1,103.3,2171.9);
H1P1V38b = add_vertex(H1P1,103.3,2206.6);
H1P1V38c = add_vertex(H1P1,342.1,2206.6);
H1P1V38d = add_vertex(H1P1,342.1,2099.5);
H1P1V38e = add_vertex(H1P1,103.3,2099.5);
H1P1V38f = add_vertex(H1P1,103.3,2136.2);

H1P1V39 = add_vertex(H1P1,98.0,2136.2);
H1P1V40 = add_vertex(H1P1,98.0,1937.7);   /* rm 531 */
H1P1V40a = add_vertex(H1P1,103.3,1937.7);
H1P1V40b = add_vertex(H1P1,103.3,1972.7);
H1P1V40c = add_vertex(H1P1,342.1,1972.7);
H1P1V40d = add_vertex(H1P1,342.1,1865.6);
H1P1V40e = add_vertex(H1P1,103.3,1865.6);
H1P1V40f = add_vertex(H1P1,103.3,1877.7);

H1P1V41 = add_vertex(H1P1,98.0,1877.7);
H1P1V42 = add_vertex(H1P1,98.0,1744.5);   /* rm 529 */
H1P1V42a = add_vertex(H1P1,103.3,1744.5);
H1P1V42b = add_vertex(H1P1,103.3,1779.5);
H1P1V42c = add_vertex(H1P1,342.1,1779.5);
H1P1V42d = add_vertex(H1P1,342.1,1672.4);
H1P1V42e = add_vertex(H1P1,103.3,1672.4);
H1P1V42f = add_vertex(H1P1,103.3,1684.5);

H1P1V43 = add_vertex(H1P1,98.0,1684.5);
H1P1V44 = add_vertex(H1P1,98.0,1522.4);   /* rm 527 */
H1P1V44a = add_vertex(H1P1,103.3,1522.4);
H1P1V44b = add_vertex(H1P1,103.3,1557.4);
```

```
H1P1V44c = add_vertex(H1P1,342.1,1557.4);
H1P1V44d = add_vertex(H1P1,342.1,1450.3);
H1P1V44e = add_vertex(H1P1,103.3,1450.3);
H1P1V44f = add_vertex(H1P1,103.3,1462.4);

H1P1V45 = add_vertex(H1P1,98.0,1462.4);
H1P1V46 = add_vertex(H1P1,98.0,1342.7);   /* rm 525 */
H1P1V46a = add_vertex(H1P1,103.3,1342.7);
H1P1V46b = add_vertex(H1P1,103.3,1377.7);
H1P1V46c = add_vertex(H1P1,342.1,1377.7);
H1P1V46d = add_vertex(H1P1,342.1,1270.6);
H1P1V46e = add_vertex(H1P1,103.3,1270.6);
H1P1V46f = add_vertex(H1P1,103.3,1307.0);

H1P1V47 = add_vertex(H1P1,98.0,1307.0);
H1P1V48 = add_vertex(H1P1,98.0,1118.8);   /* rm 523 */
H1P1V48a = add_vertex(H1P1,103.3,1118.8);
H1P1V48b = add_vertex(H1P1,103.3,1153.8);
H1P1V48c = add_vertex(H1P1,342.1,1153.8);
H1P1V48d = add_vertex(H1P1,342.1,1046.7);
H1P1V48e = add_vertex(H1P1,103.3,1046.7);
H1P1V48f = add_vertex(H1P1,103.3,1083.1);

H1P1V49 = add_vertex(H1P1,98.0,1083.1);
H1P1V50 = add_vertex(H1P1,98.0,796.1);   /* rm 521 */
H1P1V50a = add_vertex(H1P1,103.3,796.1);
H1P1V50b = add_vertex(H1P1,103.3,831.1);
H1P1V50c = add_vertex(H1P1,342.1,831.1);
H1P1V50d = add_vertex(H1P1,342.1,724.0);
H1P1V50e = add_vertex(H1P1,103.3,724.0);
H1P1V50f = add_vertex(H1P1,103.3,760.4);

H1P1V51 = add_vertex(H1P1,98.0,760.4);
H1P1V52 = add_vertex(H1P1,98.0,564.5);   /* rm 519 */
H1P1V52a = add_vertex(H1P1,103.3,564.5);
H1P1V52b = add_vertex(H1P1,103.3,599.5);
H1P1V52c = add_vertex(H1P1,342.1,599.5);
H1P1V52d = add_vertex(H1P1,342.1,492.4);
H1P1V52e = add_vertex(H1P1,103.3,492.4);
H1P1V52f = add_vertex(H1P1,103.3,528.8);

H1P1V53 = add_vertex(H1P1,98.0,528.8);
H1P1V54 = add_vertex(H1P1,98.0,413.9);   /* corners */
H1P1V55 = add_vertex(H1P1,257.9,413.9);   /* rm ? */
H1P1V55a = add_vertex(H1P1,257.9,419.2);
H1P1V55b = add_vertex(H1P1,221.7,419.2);
H1P1V55c = add_vertex(H1P1,221.7,500.0);
H1P1V55d = add_vertex(H1P1,300.0,500.0);
H1P1V55e = add_vertex(H1P1,300.0,419.2);
H1P1V55f = add_vertex(H1P1,293.9,419.2);
```

```
H1P1V56 = add_vertex(H1P1,293.9,413.9);
H1P1V57 = add_vertex(H1P1,337.5,413.9);
H1P1V58 = add_vertex(H1P1,337.5,402.6);   /* office */
H1P1V58a = add_vertex(H1P1,342.8,402.6);
H1P1V58b = add_vertex(H1P1,342.8,600.0);
H1P1V58c = add_vertex(H1P1,449.9,600.0);
H1P1V58d = add_vertex(H1P1,449.9,330.0);
H1P1V58e = add_vertex(H1P1,342.8,330.0);
H1P1V58f = add_vertex(H1P1,342.8,342.6);

H1P1V59 = add_vertex(H1P1,337.5,342.6);
H1P1V60 = add_vertex(H1P1,337.5,310.2);   /* rm 511 */
H1P1V60a = add_vertex(H1P1,342.8,310.2);
H1P1V60b = add_vertex(H1P1,342.8,315.0);
H1P1V60c = add_vertex(H1P1,449.9,315.0);
H1P1V60d = add_vertex(H1P1,449.9,0.0);
H1P1V60e = add_vertex(H1P1,342.8,0.0);
H1P1V60f = add_vertex(H1P1,342.8,274.5);

H1P1V61 = add_vertex(H1P1,337.5,274.5);
H1P1V62 = add_vertex(H1P1,337.5,267.4);
H1P1V63 = add_vertex(H1P1,306.9,267.4);   /* elev 1 (left)*/
H1P1V63a = add_vertex(H1P1,306.9,267.7);
H1P1V63b = add_vertex(H1P1,303.9,267.7);
H1P1V63c = add_vertex(H1P1,303.9,255.7);
H1P1V63d = add_vertex(H1P1,277.9,255.7);
H1P1V63e = add_vertex(H1P1,251.9,255.7);
H1P1V63f = add_vertex(H1P1,251.9,267.7);
H1P1V63g = add_vertex(H1P1,248.9,267.7);
H1P1V64 = add_vertex(H1P1,248.9,267.4);
H1P1V65 = add_vertex(H1P1,192.2,267.4);

                                             /* elev 2
*/
H1P1V65a = add_vertex(H1P1,192.2,267.7);
H1P1V65b = add_vertex(H1P1,189.2,267.7);
H1P1V65c = add_vertex(H1P1,189.2,255.7);
H1P1V65d = add_vertex(H1P1,163.2,255.7);
H1P1V65e = add_vertex(H1P1,137.2,255.7);
H1P1V65f = add_vertex(H1P1,137.2,267.7);
H1P1V65g = add_vertex(H1P1,134.2,267.7);
H1P1V66 = add_vertex(H1P1,134.2,267.4);

H1P1V67 = add_vertex(H1P1,98.0,267.4);
H1P1V68 = add_vertex(H1P1,98.0,100.0);   /* stairwell */
H1P1V68a = add_vertex(H1P1,103.3,100.0);
H1P1V68b = add_vertex(H1P1,103.3,125.0);
H1P1V68c = add_vertex(H1P1,150.0,125.0);
H1P1V68d = add_vertex(H1P1,150.0,40.0);
H1P1V68e = add_vertex(H1P1,103.3,40.0);
H1P1V68f = add_vertex(H1P1,103.3,64.3);
```

```
H1P1V69 = add_vertex(H1P1,98.0,64.3);
H1P1V70 = add_vertex(H1P1,98.0,0.0);


H1P2=add_pg(H1,102.0,0,1);  /*main ceiling*/
H1P2V1 = add_vertex(H1P2,0.0,0.0);
H1P2V2 = add_vertex(H1P2,0.0,2351.2);
H1P2V3 = add_vertex(H1P2,98.0,2351.2);
H1P2V4 = add_vertex(H1P2,98.0,0.0);


H1P3=add_pg(H1,113.3,0,1);  /*elev ceiling*/
H1P3V1 = add_vertex(H1P3,98.0,267.4);
H1P3V2 = add_vertex(H1P3,98.0,413.9);
H1P3V3 = add_vertex(H1P3,337.5,413.9);
H1P3V4 = add_vertex(H1P3,337.5,267.4);


H1P4 = add_pg(H1,84.0,0,1);   /*rm 506 door jam ceiling*/
H1P4V1= add_vertex(H1P4,0.0,239.5);
H1P4V2= add_vertex(H1P4,-5.3,239.5);
H1P4V3= add_vertex(H1P4,-5.3,275.2);
H1P4V4= add_vertex(H1P4,0.0,275.2);
H1P5 = add_pg(H1,84.0,0,1);    /*rm 510 door jam ceiling*/
H1P5V1= add_vertex(H1P5,0.0,713.7);
H1P5V2= add_vertex(H1P5,-5.3,713.7);
H1P5V3= add_vertex(H1P5,-5.3,749.4);
H1P5V4= add_vertex(H1P5,0.0,749.4);
H1P6 = add_pg(H1,84.0,0,1);    /*rm 512 door jam ceiling*/
H1P6V1= add_vertex(H1P6,0.0,825.9);
H1P6V2= add_vertex(H1P6,-5.3,825.9);
H1P6V3= add_vertex(H1P6,-5.3,861.6);
H1P6V4= add_vertex(H1P6,0.0,861.6);
H1P7 = add_pg(H1,84.0,0,1);    /*rm 514 door jam ceiling*/
H1P7V1= add_vertex(H1P7,0.0,937.5);
H1P7V2= add_vertex(H1P7,-5.3,937.5);
H1P7V3= add_vertex(H1P7,-5.3,973.2);
H1P7V4= add_vertex(H1P7,0.0,973.2);
H1P8 = add_pg(H1,84.0,0,1);    /*rm 516 door jam ceiling*/
H1P8V1= add_vertex(H1P8,0.0,1049.7);
H1P8V2= add_vertex(H1P8,-5.3,1049.7);
H1P8V3= add_vertex(H1P8,-5.3,1085.4);
H1P8V4= add_vertex(H1P8,0.0,1085.4);
H1P9 = add_pg(H1,84.0,0,1);    /*rm 518 door jam ceiling*/
H1P9V1= add_vertex(H1P9,0.0,1161.7);
H1P9V2= add_vertex(H1P9,-5.3,1161.7);
H1P9V3= add_vertex(H1P9,-5.3,1197.4);
H1P9V4= add_vertex(H1P9,0.0,1197.4);
H1P10 = add_pg(H1,84.0,0,1);    /*rm 520 door jam ceiling*/
H1P10V1= add_vertex(H1P10,0.0,1273.4);
H1P10V2= add_vertex(H1P10,-5.3,1273.4);
H1P10V3= add_vertex(H1P10,-5.3,1309.1);
H1P10V4= add_vertex(H1P10,0.0,1309.1);
H1P11 = add_pg(H1,84.0,0,1);    /*rm 522R door jam ceiling*/
```

```
H1P11V1= add_vertex(H1P11,0.0,1429.6);
H1P11V2= add_vertex(H1P11,-5.3,1429.6);
H1P11V3= add_vertex(H1P11,-5.3,1461.3);
H1P11V4= add_vertex(H1P11,0.0,1461.3);
H1P12 = add_pg(H1,84.0,0,1);    /*rm FD 1 door jam ceiling*/
H1P12V1= add_vertex(H1P12,0.0,1488.0);
H1P12V2= add_vertex(H1P12,-5.5,1488.0);
H1P12V3= add_vertex(H1P12,-5.5,1560.0);
H1P12V4= add_vertex(H1P12,0.0,1560.0);
H1P13 = add_pg(H1,84.0,0,1);    /*rm 524 door jam ceiling*/
H1P13V1= add_vertex(H1P13,0.0,1583.3);
H1P13V2= add_vertex(H1P13,-5.3,1583.3);
H1P13V3= add_vertex(H1P13,-5.3,1619.0);
H1P13V4= add_vertex(H1P13,0.0,1619.0);
H1P14 = add_pg(H1,84.0,0,1);    /* 526R ceiling*/
H1P14V1= add_vertex(H1P14,0.0,1754.5);
H1P14V2= add_vertex(H1P14,-5.3,1754.5);
H1P14V3= add_vertex(H1P14,-5.3,1786.2);
H1P14V4= add_vertex(H1P14,0.0,1786.2);
H1P15 = add_pg(H1,84.0,0,1);    /*rm 528A door jam ceiling*/
H1P15V1= add_vertex(H1P15,0.0,1836.4);
H1P15V2= add_vertex(H1P15,-5.3,1836.4);
H1P15V3= add_vertex(H1P15,-5.3,1872.1);
H1P15V4= add_vertex(H1P15,0.0,1872.1);
H1P16 = add_pg(H1,84.0,0,1);    /*rm 528B door jam ceiling*/
H1P16V1= add_vertex(H1P16,0.0,1919.1);
H1P16V2= add_vertex(H1P16,-5.3,1919.1);
H1P16V3= add_vertex(H1P16,-5.3,1954.8);
H1P16V4= add_vertex(H1P16,0.0,1954.8);
H1P17 = add_pg(H1,84.0,0,1);    /*rm 530A door jam ceiling*/
H1P17V1= add_vertex(H1P17,0.0,2030.4);
H1P17V2= add_vertex(H1P17,-5.3,2030.4);
H1P17V3= add_vertex(H1P17,-5.3,2066.1);
H1P17V4= add_vertex(H1P17,0.0,2066.1);
H1P18 = add_pg(H1,84.0,0,1);    /*rm 530B door jam ceiling*/
H1P18V1= add_vertex(H1P18,0.0,2195.1);
H1P18V2= add_vertex(H1P18,-5.3,2195.1);
H1P18V3= add_vertex(H1P18,-5.3,2230.8);
H1P18V4= add_vertex(H1P18,0.0,2230.8);
H1P19 = add_pg(H1,84.0,0,1);    /*rm 530C door jam ceiling*/
H1P19V1= add_vertex(H1P19,0.0,2253.8);
H1P19V2= add_vertex(H1P19,-5.3,2253.8);
H1P19V3= add_vertex(H1P19,-5.3,2289.5);
H1P19V4= add_vertex(H1P19,0.0,2289.5);
H1P20 = add_pg(H1,84.0,0,1);    /*rm 421 door jam ceiling*/
H1P20V1= add_vertex(H1P20,98.0,2171.9);
H1P20V2= add_vertex(H1P20,103.3,2171.9);
H1P20V3= add_vertex(H1P20,103.3,2136.2);
H1P20V4= add_vertex(H1P20,98.0,2136.2);
H1P21 = add_pg(H1,84.0,0,1);    /*rm 531 door jam ceiling*/
H1P21V1= add_vertex(H1P21,98.0,1937.7);
```

```
H1P21V2= add_vertex(H1P21,103.3,1937.7);
H1P21V3= add_vertex(H1P21,103.3,1877.7);
H1P21V4= add_vertex(H1P21,98.0,1877.7);
H1P22 = add_pg(H1,84.0,0,1);    /*rm 529 door jam ceiling*/
H1P22V1= add_vertex(H1P22,98.0,1744.5);
H1P22V2= add_vertex(H1P22,103.3,1744.5);
H1P22V3= add_vertex(H1P22,103.3,1684.5);
H1P22V4= add_vertex(H1P22,98.0,1684.5);
H1P23 = add_pg(H1,84.0,0,1);    /*rm 527 door jam ceiling*/
H1P23V1= add_vertex(H1P23,98.0,1522.4);
H1P23V2= add_vertex(H1P23,103.3,1522.4);
H1P23V3= add_vertex(H1P23,103.3,1462.4);
H1P23V4= add_vertex(H1P23,98.0,1462.4);
H1P24 = add_pg(H1,84.0,0,1);    /*rm 525 door jam ceiling*/
H1P24V1= add_vertex(H1P24,98.0,1342.7);
H1P24V2= add_vertex(H1P24,103.3,1342.7);
H1P24V3= add_vertex(H1P24,103.0,1307.0);
H1P24V4= add_vertex(H1P24,98.0,1307.0);
H1P25 = add_pg(H1,84.0,0,1);    /*rm 523 door jam ceiling*/
H1P25V1= add_vertex(H1P25,98.0,1118.8);
H1P25V2= add_vertex(H1P25,103.3,1118.8);
H1P25V3= add_vertex(H1P25,103.3,1083.1);
H1P25V4= add_vertex(H1P25,98.0,1083.1);
H1P26 = add_pg(H1,84.0,0,1);    /*rm 521 door jam ceiling*/
H1P26V1= add_vertex(H1P26,98.0,796.1);
H1P26V2= add_vertex(H1P26,103.3,796.1);
H1P26V3= add_vertex(H1P26,103.3,760.4);
H1P26V4= add_vertex(H1P26,98.0,760.4);
H1P27 = add_pg(H1,84.0,0,1);    /*rm 519 door jam ceiling*/
H1P27V1= add_vertex(H1P27,98.0,564.5);
H1P27V2= add_vertex(H1P27,103.3,564.5);
H1P27V3= add_vertex(H1P27,103.3,528.8);
H1P27V4= add_vertex(H1P27,98.0,528.8);
H1P28 = add_pg(H1,84.0,0,1);    /*rm ? door jam ceiling*/
H1P28V1= add_vertex(H1P28,257.9,413.9);
H1P28V2= add_vertex(H1P28,257.9,419.2);
H1P28V3= add_vertex(H1P28,293.9,419.2);
H1P28V4= add_vertex(H1P28,293.9,413.9);
H1P29 = add_pg(H1,84.0,0,1);    /* office door jam ceiling*/
H1P29V1= add_vertex(H1P29,337.5,402.6);
H1P29V2= add_vertex(H1P29,342.8,402.6);
H1P29V3= add_vertex(H1P29,342.8,342.6);
H1P29V4= add_vertex(H1P29,337.5,342.6);
H1P30 = add_pg(H1,84.0,0,1);    /* rm 511 door jam ceiling*/
H1P30V1= add_vertex(H1P30,337.5,310.2);
H1P30V2= add_vertex(H1P30,342.8,310.2);
H1P30V3= add_vertex(H1P30,342.8,274.5);
H1P30V4= add_vertex(H1P30,337.5,274.5);

H1P31 = add_pg(H1,83.8,0,1);    /* elev 1 door jam ceiling*/
H1P31V1= add_vertex(H1P31,303.9,267.7);
```

```
H1P31V2= add_vertex(H1P31,303.9,255.7);
H1P31V3= add_vertex(H1P31,277.9,255.7);
H1P31V4= add_vertex(H1P31,251.9,255.7);
H1P31V5= add_vertex(H1P31,251.9,267.7);
H1P32 = add_pg(H1,83.8,0,1);    /* elev 2 door jam ceiling*/
H1P32V1= add_vertex(H1P32,189.2,267.7);
H1P32V2= add_vertex(H1P32,189.2,255.7);
H1P32V3= add_vertex(H1P32,163.2,255.7);
H1P32V4= add_vertex(H1P32,137.2,255.7);
H1P32V5= add_vertex(H1P32,137.2,267.7);


H1P63 = add_pg(H1,86.8,0,1);    /* elev 1 ceiling*/
H1P63V1= add_vertex(H1P63,306.9,267.4);
H1P63V2= add_vertex(H1P63,306.9,267.7);
H1P63V3= add_vertex(H1P63,248.9,267.7);
H1P63V4= add_vertex(H1P63,248.9,267.4);


H1P64 = add_pg(H1,86.8,0,1);    /* elev 2 ceiling*/
H1P64V1= add_vertex(H1P64,192.2,267.4);
H1P64V2= add_vertex(H1P64,192.2,267.7);
H1P64V3= add_vertex(H1P64,134.2,267.7);
H1P64V4= add_vertex(H1P64,134.2,267.4);

H1P33 = add_pg(H1,84.0,0,1);    /* stairwell door jam ceiling*/
H1P33V1= add_vertex(H1P33,98.0,100.0);
H1P33V2= add_vertex(H1P33,103.3,100.0);
H1P33V3= add_vertex(H1P33,103.3,64.3);
H1P33V4= add_vertex(H1P33,98.0,64.3);

H1P34 = add_pg(H1,144.0,0,1);    /*rm 506 ceiling*/
H1P34V1= add_vertex(H1P34,-5.3,203.3);
H1P34V2= add_vertex(H1P34,-244.1,203.3);
H1P34V3= add_vertex(H1P34,-244.1,309.4);
H1P34V4= add_vertex(H1P34,-5.3,309.4);
H1P35 = add_pg(H1,144.0,0,1);    /*rm 510 ceiling*/
H1P35V1= add_vertex(H1P35,-5.3,677.5);
H1P35V2= add_vertex(H1P35,-244.1,677.5);
H1P35V3= add_vertex(H1P35,-244.1,783.6);
H1P35V4= add_vertex(H1P35,-5.3,783.6);
H1P36 = add_pg(H1,144.0,0,1);    /*rm 512 ceiling*/
H1P36V1= add_vertex(H1P36,-5.3,789.7);
H1P36V2= add_vertex(H1P36,-244.1,789.7);
H1P36V3= add_vertex(H1P36,-244.1,895.8);
H1P36V4= add_vertex(H1P36,-5.3,895.8);
H1P37 = add_pg(H1,144.0,0,1);    /*rm 514 ceiling*/
H1P37V1= add_vertex(H1P37,-5.3,901.3);
H1P37V2= add_vertex(H1P37,-244.1,901.3);
H1P37V3= add_vertex(H1P37,-244.1,1007.4);
H1P37V4= add_vertex(H1P37,-5.3,1007.4);
H1P38 = add_pg(H1,144.0,0,1);    /*rm 516 ceiling*/
```

```
H1P38V1= rdd_vertex(H1P38,-5.3,1013.5);
H1P38V2= add_vertex(H1P38,-244.1,1013.5);
H1P38V3= add_vertex(H1P38,-244.1,1119.6);
H1P38V4= add_vertex(H1P38,-5.3,1119.6);
H1P39 = add_pg(H1,144.0,0,1);    /*rm 518 ceiling*/
H1P39V1= add_vertex(H1P39,-5.3,1125.5);
H1P39V2= add_vertex(H1P39,-244.1,1125.5);
H1P39V3= add_vertex(H1P39,-244.1,1231.6);
H1P39V4= add_vertex(H1P39,-5.3,1231.6);
H1P40 = add_pg(H1,144.0,0,1);    /*rm 520 ceiling*/
H1P40V1= add_vertex(H1P40,-5.3,1237.2);
H1P40V2= add_vertex(H1P40,-244.1,1237.2);
H1P40V3= add_vertex(H1P40,-244.1,1343.3);
H1P40V4= add_vertex(H1P40,-5.3,1343.3);
H1P41 = add_pg(H1,144 0,0,1);    /*rm 522R ceiling*/
H1P41V1= add_vertex(H1P41,-5.3,1393.4);
H1P41V2= add_vertex(H1P41,-244.1,1393.4);
H1P41V3= add_vertex(H1P41,-244.1,1499.5);
H1P41V4= add_vertex(H1P41,-5.3,1499.5);
H1P42 = add_pg(H1,144.0,0,1);    /* FD1 ceiling*/
H1P42V1= add_vertex(H1P42,-5.5,1486.0);
H1P42V2= add_vertex(H1P42,-50.0,1486.0);
H1P42V3= add_vertex(H1P42,-50.0,1562.0);
H1P42V4= add_vertex(H1P42,-5.5,1562.0);
H1P43 = add_pg(H1,144.0,0,1);    /*rm 524 ceiling*/
H1P43V1= add_vertex(H1P43,-5.3,1547.1);
H1P43V2= add_vertex(H1P43,-244.1,1547.1);
H1P43V3= add_vertex(H1P43,-244.1,1653.2);
H1P43V4= add_vertex(H1P43,-5.3,1653.2);
H1P44 = add_pg(H1,84.0,0,1);     /* water fountain ceiling*/
H1P44V1= add_vertex(H1P44,0.0,1650.4);
H1P44V2= add_vertex(H1P44,-30.0,1650.4);
H1P44V3= add_vertex(H1P44,-30.0,1684.5);
H1P44V4= add_vertex(H1P44,0.0,1684.5);
H1P45 = add_pg(H1,144.0,0,1);    /*rm 526R ceiling*/
H1P45V1= add_vertex(H1P45,-5.3,1718.3);
H1P45V2= add_vertex(H1P45,-244.1,1718.3);
H1P45V3= add_vertex(H1P45,-244.1,1790.0);
H1P45V4= add_vertex(H1P45,-5.3,1790.0);
H1P46 = add_pg(H1,144.0,0,1);    /*rm 528A ceiling*/
H1P46V1= add_vertex(H1P46,-5.3,1800.2);
H1P46V2= add_vertex(H1P46,-244.1,1800.2);
H1P46V3= add_vertex(H1P46,-244.1,1875.0);
H1P46V4= add_vertex(H1P46,-5.3,1875.0);
H1P47 = add_pg(H1,144.0,0,1);    /*rm 528B ceiling*/
H1P47V1= add_vertex(H1P47,-5.3,1882.9);
H1P47V2= add_vertex(H1P47,-244.1,1882.9);
H1P47V3= add_vertex(H1P47,-244.1,1989.0);
H1P47V4= add_vertex(H1P47,-5.3,1989.0);
H1P48 = add_pg(H1,144.0,0,1);    /*rm 530A ceiling*/
H1P48V1= add_vertex(H1P48,-5.3,1994.2);
```

```
H1P48V2= add_vertex(H1P48,-244.1,1994.2);
H1P48V3= add_vertex(H1P48,-244.1,2100.3);
H1P48V4= add_vertex(H1P48,-5.3,2100.3);
H1P49 = add_pg(H1,144.0,0,1);    /*rm 530B ceiling*/
H1P49V1= add_vertex(H1P49,-5.3,2158.8);
H1P49V2= add_vertex(H1P49,-244.1,2158.8);
H1P49V3= add_vertex(H1P49,-244.1,2250.0);
H1P49V4= add_vertex(H1P49,-5.3,2250.0);
H1P50 = add_pg(H1,144.0,0,1);    /*rm 530C ceiling*/
H1P50V1= add_vertex(H1P50,-5.3,2251.0);
H1P50V2= add_vertex(H1P50,-244.1,2251.0);
H1P50V3= add_vertex(H1P50,-244.1,2350.0);
H1P50V4= add_vertex(H1P50.-5.3,2350.0);

/* following ceilings are incorrect based on 35° to either side of door*/

H1P51 = add_pg(H1,144.0,0,1);    /*rm 421 ceiling*/
H1P51V1= add_vertex(H1P51,103.3,2206.6);
H1P51V2= add_vertex(H1P51,342.1,2206.6);
H1P51V3= add_vertex(H1P51,342.1,2099.5);
H1P51V4= add_vertex(H1P51,103.3,2099.5);
H1P52 = add_pg(H1,144.0,0,1);    /*rm 531 ceiling*/
H1P52V1= add_vertex(H1P52,103.3,1972.7);
H1P52V2= add_vertex(H1P52,342.1,1972.7);
H1P52V3= add_vertex(H1P52,342.1,1865.6);
H1P52V4= add_vertex(H1P52,103.3,1865.6);
H1P53 = add_pg(H1,144.0,0,1);    /*rm 529 ceiling*/
H1P53V1= add_vertex(H1P53,103.3,1779.5);
H1P53V2= add_vertex(H1P53,342.1,1779.5);
H1P53V3= add_vertex(H1P53,342.1,1672.4);
H1P53V4= add_vertex(H1P53,103.3,1672.4);
H1P54 = add_pg(H1,144.0,0,1);    /*rm 527 ceiling*/
H1P54V1= add_vertex(H1P54,103.3,1557.4);
H1P54V2= add_vertex(H1P54,342.1,1557.4);
H1P54V3= add_vertex(H1P54,342.1,1450.3);
H1P54V4= add_vertex(H1P54,103.3,1450.3);
H1P55 = add_pg(H1,144.0,0,1);    /*rm 525 ceiling*/
H1P55V1= add_vertex(H1P55,103.3,1377.7);
H1P55V2= add_vertex(H1P55,342.1,1377.7);
H1P55V3= add_vertex(H1P55,342.1,1270.6);
H1P55V4= add_vertex(H1P55,103.3,1270.6);
H1P56 = add_pg(H1,144.0,0,1);    /*rm 523 ceiling*/
H1P56V1= add_vertex(H1P56,103.3,1153.8);
H1P56V2= add_vertex(H1P56,342.1,1153.8);
H1P56V3= add_vertex(H1P56,342.1,1046.7);
H1P56V4= add_vertex(H1P56,103.3,1046.7);
H1P57 = add_pg(H1,144.0,0,1);    /*rm 521 ceiling*/
H1P57V1= add_vertex(H1P57,103.3,8?1.1);
H1P57V2= add_vertex(H1P57,342.1,831.1);
H1P57V3= add_vertex(H1P57,342.1,724.0);
H1P57V4= add_vertex(H1P57,103.3,724.0);
```

```
H1P58 = add_pg(H1,144.0,0,1);   /*rm 519 ceiling*/
H1P58V1= add_vertex(H1P58,103.3,599.5);
H1P58V2= add_vertex(H1P58,342.1,599.5);
H1P58V3= add_vertex(H1P58,342.1,492.4);
H1P58V4= add_vertex(H1P58,103.3,492.4);
H1P59 = add_pg(H1,144.0,0,1);   /*rm ? ceiling*/
H1P59V1= add_vertex(H1P59,221.7,419.2);
H1P59V2= add_vertex(H1P59,221.7,500.0);
H1P59V3= add_vertex(H1P59,300.0,500.0);
H1P59V4= add_vertex(H1P59,300.0,419.2);
H1P60 = add_pg(H1,144.0,0,1);   /* office ceiling*/
H1P60V1= add_vertex(H1P60,342.8,600.0);
H1P60V2= add_vertex(H1P60,449.9,600.0);
H1P60V3= add_vertex(H1P60,449.9,330.0);
H1P60V4= add_vertex(H1P60,342.8,330.0);
H1P61 = add_pg(H1,144.0,0,1);   /*rm 511 ceiling*/
H1P61V1= add_vertex(H1P61,342.8,315.0);
H1P61V2= add_vertex(H1P61,449.9,315.0);
H1P61V3= add_vertex(H1P61,449.9,0.0);
H1P61V4= add_vertex(H1P61,342.8,0.0);
H1P62 = add_pg(H1,144.0,0,1);   /*rm stairwell ceiling*/
H1P62V1= add_vertex(H1P62,103.3,125.0);
H1P62V2= add_vertex(H1P62,150.0,125.0);
H1P62V3= add_vertex(H1P62,150.0,40.0);
H1P62V4= add_vertex(H1P62,103.3,40.0);

/* Don't forget to add the ceiling associations or else we can't tell
   how high each section of the hallway is*/

add_ceiling(H1P1,H1P2);
add_ceiling(H1P1,H1P3);
add_ceiling(H1P1,H1P4);
add_ceiling(H1P1,H1P5);
add_ceiling(H1P1,H1P6);
add_ceiling(H1P1,H1P7);
add_ceiling(H1P1,H1P8);
add_ceiling(H1P1,H1P9);
add_ceiling(H1P1,H1P10);
add_ceiling(H1P1,H1P11);
add_ceiling(H1P1,H1P12);
add_ceiling(H1P1,H1P13);
add_ceiling(H1P1,H1P14);
add_ceiling(H1P1,H1P15);
add_ceiling(H1P1,H1P16);
add_ceiling(H1P1,H1P17);
add_ceiling(H1P1,H1P18);
add_ceiling(H1P1,H1P19);
add_ceiling(H1P1,H1P20);
add_ceiling(H1P1,H1P21);
add_ceiling(H1P1,H1P22);
add_ceiling(H1P1,H1P23);
```

```
        add_ceiling(H1P1,H1P24);
        add_ceiling(H1P1,H1P25);
        add_ceiling(H1P1,H1P26);
        add_ceiling(H1P1,H1P27);
        add_ceiling(H1P1,H1P28);
        add_ceiling(H1P1,H1P29);
        add_ceiling(H1P1,H1P30);
        add_ceiling(H1P1,H1P31);
        add_ceiling(H1P1,H1P32);
        add_ceiling(H1P1,H1P33);
        add_ceiling(H1P1,H1P34);
        add_ceiling(H1P1,H1P35);
        add_ceiling(H1P1,H1P36);
        add_ceiling(H1P1,H1P37);
        add_ceiling(H1P1,H1P38);
        add_ceiling(H1P1,H1P39);
        add_ceiling(H1P1,H1P40);
        add_ceiling(H1P1,H1P41);
        add_ceiling(H1P1,H1P42);
        add_ceiling(H1P1,H1P43);
        add_ceiling(H1P1,H1P44);
        add_ceiling(H1P1,H1P45);
        add_ceiling(H1P1,H1P46);
        add_ceiling(H1P1,H1P47);
        add_ceiling(H1P1,H1P48);
        add_ceiling(H1P1,H1P49);
        add_ceiling(H1P1,H1P50);
        add_ceiling(H1P1,H1P51);
        add_ceiling(H1P1,H1P52);
        add_ceiling(H1P1,H1P53);
        add_ceiling(H1P1,H1P54);
        add_ceiling(H1P1,H1P55);
        add_ceiling(H1P1,H1P56);
        add_ceiling(H1P1,H1P57);
        add_ceiling(H1P1,H1P58);
        add_ceiling(H1P1,H1P59);
        add_ceiling(H1P1,H1P60);
        add_ceiling(H1P1,H1P61);
        add_ceiling(H1P1,H1P62);
        add_ceiling(H1P1,H1P63);
        add_ceiling(H1P1,H1P64);

/* Vertical edges must alway be explicitly added */

add_edge(H1P1V1,H1P2V1);
add_edge(H1P1V2,H1P4V1);     /*link up vert edges of room 506*/
add_edge(H1P1V2a,H1P4V2);
add_edge(H1P1V2b,H1P34V1);
add_edge(H1P1V2c,H1P34V2);
add_edge(H1P1V2d,H1P34V3);
add_edge(H1P1V2e,H1P34V4);
```

```
add_edge(H1P1V2f,H1P4V3);
add_edge(H1P1V3,H1P4V4);
add_edge(H1P1V4,H1P5V1);      /*link up vert edges of room 510*/
add_edge(H1P1V4a,H1P5V2);
add_edge(H1P1V4b,H1P35V1);
add_edge(H1P1V4c,H1P35V2);
add_edge(H1P1V4d,H1P35V3);
add_edge(H1P1V4e,H1P35V4);
add_edge(H1P1V4f,H1P5V3);
add_edge(H1P1V5,H1P5V4);
add_edge(H1P1V6,H1P6V1);      /*link up vert edges of room 512 */
add_edge(H1P1V6a,H1P6V2);
add_edge(H1P1V6b,H1P36V1);
add_edge(H1P1V6c,H1P36V2);
add_edge(H1P1V6d,H1P36V3);
add_edge(H1P1V6e,H1P36V4);
add_edge(H1P1V6f,H1P6V3);
add_edge(H1P1V7,H1P6V4);
add_edge(H1P1V8,H1P7V1);      /*link up vert edges of room 514*/
add_edge(H1P1V8a,H1P7V2);
add_edge(H1P1V8b,H1P37V1);
add_edge(H1P1V8c,H1P37V2);
add_edge(H1P1V8d,H1P37V3);
add_edge(H1P1V8e,H1P37V4);
add_edge(H1P1V8f,H1P7V3);
add_edge(H1P1V9,H1P7V4);
add_edge(H1P1V10,H1P8V1);      /*link up vert edges of room 516*/
add_edge(H1P1V10a,H1P8V2);
add_edge(H1P1V10b,H1P38V1);
add_edge(H1P1V10c,H1P38V2);
add_edge(H1P1V10d,H1P38V3);
add_edge(H1P1V10e,H1P38V4);
add_edge(H1P1V10f,H1P8V3);
add_edge(H1P1V11,H1P8V4);
add_edge(H1P1V12,H1P9V1);      /*link up vert edges of room 518*/
add_edge(H1P1V12a,H1P9V2);
add_edge(H1P1V12b,H1P39V1);
add_edge(H1P1V12c,H1P39V2);
add_edge(H1P1V12d,H1P39V3);
add_edge(H1P1V12e,H1P39V4);
add_edge(H1P1V12f,H1P9V3);
add_edge(H1P1V13,H1P9V4);
add_edge(H1P1V14,H1P10V1);      /*link up vert edges of room 520*/
add_edge(H1P1V14a,H1P10V2);
add_edge(H1P1V14b,H1P40V1);
add_edge(H1P1V14c,H1P40V2);
add_edge(H1P1V14d,H1P40V3);
add_edge(H1P1V14e,H1P40V4);
add_edge(H1P1V14f,H1P10V3);
add_edge(H1P1V15,H1P10V4);
add_edge(H1P1V16,H1P11V1);      /*link up vert edges of room 522R*/
```

```
add_edge(H1P1V16a,H1P11V2);
add_edge(H1P1V16b,H1P41V1);
add_edge(H1P1V16c,H1P41V2);
add_edge(H1P1V16d,H1P41V3);
add_edge(H1P1V16e,H1P41V4);
add_edge(H1P1V16f,H1P11V3);
add_edge(H1P1V17,H1P11V4);
add_edge(H1P1V18,H1P12V1);     /*link up vert edges of room FD1*/
add_edge(H1P1V18a,H1P12V2);
add_edge(H1P1V18b,H1P42V1);
add_edge(H1P1V18c,H1P42V2);
add_edge(H1P1V18d,H1P42V3);
add_edge(H1P1V18e,H1P42V4);
add_edge(H1P1V18f,H1P12V3);
add_edge(H1P1V19,H1P12V4);
add_edge(H1P1V20,H1P13V1);     /*link up vert edges of room 524*/
add_edge(H1P1V20a,H1P13V2);
add_edge(H1P1V20b,H1P43V1);
add_edge(H1P1V20c,H1P43V2);
add_edge(H1P1V20d,H1P43V3);
add_edge(H1P1V20e,H1P43V4);
add_edge(H1P1V20f,H1P13V3);
add_edge(H1P1V21,H1P13V4);
add_edge(H1P1V22,H1P44V1);     /*link up vert edges of water fountain*/
add_edge(H1P1V22a,H1P44V2);
add_edge(H1P1V22b,H1P44V3);
add_edge(H1P1V23,H1P44V4);
add_edge(H1P1V24,H1P14V1);     /*link up vert edges of room 526R*/
add_edge(H1P1V24a,H1P14V2);
add_edge(H1P1V24b,H1P45V1);
add_edge(H1P1V24c,H1P45V2);
add_edge(H1P1V24d,H1P45V3);
add_edge(H1P1V24e,H1P45V4);
add_edge(H1P1V24f,H1P14V3);
add_edge(H1P1V25,H1P14V4);
add_edge(H1P1V26,H1P15V1);     /*link up vert edges of room 528A*/
add_edge(H1P1V26a,H1P15V2);
add_edge(H1P1V26b,H1P46V1);
add_edge(H1P1V26c,H1P46V2);
add_edge(H1P1V26d,H1P46V3);
add_edge(H1P1V26e,H1P46V4);
add_edge(H1P1V26f,H1P15V3);
add_edge(H1P1V27,H1P15V4);
add_edge(H1P1V28,H1P16V1);     /*link up vert edges of room 528B*/
add_edge(H1P1V28a,H1P16V2);
add_edge(H1P1V28b,H1P47V1);
add_edge(H1P1V28c,H1P47V2);
add_edge(H1P1V28d,H1P47V3);
add_edge(H1P1V28e,H1P47V4);
add_edge(H1P1V28f,H1P16V3);
add_edge(H1P1V29,H1P16V4);
```

```
add_edge(H1P1V30,H1P17V1);    /*link up vert edges of room 530A*/
add_edge(H1P1V30a,H1P17V2);
add_edge(H1P1V30b,H1P48V1);
add_edge(H1P1V30c,H1P48V2);
add_edge(H1P1V30d,H1P48V3);
add_edge(H1P1V30e,H1P48V4);
add_edge(H1P1V30f,H1P17V3);
add_edge(H1P1V31,H1P17V4);
add_edge(H1P1V32,H1P18V1);    /*link up vert edges of room 530B*/
add_edge(H1P1V32a,H1P18V2);
add_edge(H1P1V32b,H1P49V1);
add_edge(H1P1V32c,H1P49V2);
add_edge(H1P1V32d,H1P49V3);
add_edge(H1P1V32e,H1P49V4);
add_edge(H1P1V32f,H1P18V3);
add_edge(H1P1V33,H1P18V4);
add_edge(H1P1V34,H1P19V1);    /*link up vert edges of room 530C*/
add_edge(H1P1V34a,H1P19V2);
add_edge(H1P1V34b,H1P50V1);
add_edge(H1P1V34c,H1P50V2);
add_edge(H1P1V34d,H1P50V3);
add_edge(H1P1V34e,H1P50V4);
add_edge(H1P1V34f,H1P19V3);
add_edge(H1P1V35,H1P19V4);
add_edge(H1P1V36,H1P2V2);    /*corner*/
add_edge(H1P1V37,H1P2V3);     /*corner*/
add_edge(H1P1V38,H1P20V1);    /*link up vert edges of room 421*/
add_edge(H1P1V38a,H1P20V2);
add_edge(H1P1V38b,H1P51V1);
add_edge(H1P1V38c,H1P51V2);
add_edge(H1P1V38d,H1P51V3);
add_edge(H1P1V38e,H1P51V4);
add_edge(H1P1V38f,H1P20V3);
add_edge(H1P1V39,H1P20V4);
add_edge(H1P1V40,H1P21V1);    /*link up vert edges of room 531*/
add_edge(H1P1V40a,H1P21V2);
add_edge(H1P1V40b,H1P52V1);
add_edge(H1P1V40c,H1P52V2);
add_edge(H1P1V40d,H1P52V3);
add_edge(H1P1V40e,H1P52V4);
add_edge(H1P1V40f,H1P21V3);
add_edge(H1P1V41,H1P21V4);
add_edge(H1P1V42,H1P22V1);    /*link up vert edges of room 529*/
add_edge(H1P1V42a,H1P22V2);
add_edge(H1P1V42b,H1P53V1);
add_edge(H1P1V42c,H1P53V2);
add_edge(H1P1V42d,H1P53V3);
add_edge(H1P1V42e,H1P53V4);
add_edge(H1P1V42f,H1P22V3);
add_edge(H1P1V43,H1P22V4);
add_edge(H1P1V44,H1P23V1);    /*link up vert edges of room 527*/
```

```
add_edge(H1P1V44a,H1P23V2);
add_edge(H1P1V44b,H1P54V1);
add_edge(H1P1V44c,H1P54V2);
add_edge(H1P1V44d,H1P54V3);
add_edge(H1P1V44e,H1P54V4);
add_edge(H1P1V44f,H1P23V3);
add_edge(H1P1V45,H1P23V4);
add_edge(H1P1V46,H1P24V1);     /*link up vert edges of room 525*/
add_edge(H1P1V46a,H1P24V2);
add_edge(H1P1V46b,H1P55V1);
add_edge(H1P1V46c,H1P55V2);
add_edge(H1P1V46d,H1P55V3);
add_edge(H1P1V46e,H1P55V4);
add_edge(H1P1V46f,H1P24V3);
add_edge(H1P1V47,H1P24V4);
add_edge(H1P1V48,H1P25V1);     /*link up vert edges of room 523*/
add_edge(H1P1V48a,H1P25V2);
add_edge(H1P1V48b,H1P56V1);
add_edge(H1P1V48c,H1P56V2);
add_edge(H1P1V48d,H1P56V3);
add_edge(H1P1V48e,H1P56V4);
add_edge(H1P1V48f,H1P25V3);
add_edge(H1P1V49,H1P25V4);
add_edge(H1P1V50,H1P26V1);     /*link up vert edges of room 521*/
add_edge(H1P1V50a,H1P26V2);
add_edge(H1P1V50b,H1P57V1);
add_edge(H1P1V50c,H1P57V2);
add_edge(H1P1V50d,H1P57V3);
add_edge(H1P1V50e,H1P57V4);
add_edge(H1P1V50f,H1P26V3);
add_edge(H1P1V51,H1P26V4);
add_edge(H1P1V52,H1P27V1);     /*link up vert edges of room 519*/
add_edge(H1P1V52a,H1P27V2);
add_edge(H1P1V52b,H1P58V1);
add_edge(H1P1V52c,H1P58V2);
add_edge(H1P1V52d,H1P58V3);
add_edge(H1P1V52e,H1P58V4);
add_edge(H1P1V52f,H1P27V3);
add_edge(H1P1V53,H1P27V4);
add_edge(H1P1V54,H1P3V2);    /*corner*/
add_edge(H1P1V55,H1P28V1);    /*link up vert edges of room ?*/
add_edge(H1P1V55a,H1P28V2);
add_edge(H1P1V55b,H1P59V1);
add_edge(H1P1V55c,H1P59V2);
add_edge(H1P1V55d,H1P59V3);
add_edge(H1P1V55e,H1P59V4);
add_edge(H1P1V55f,H1P28V3);
add_edge(H1P1V56,H1P28V4);
add_edge(H1P1V57,H1P3V3);    /*corner*/
add_edge(H1P1V58,H1P29V1);    /*link up vert edges of office 515 */
add_edge(H1P1V58a,H1P29V2);
```

```
add_edge(H1P1V58b,H1P60V1);
add_edge(H1P1V58c,H1P60V2);
add_edge(H1P1V58d,H1P60V3);
add_edge(H1P1V58e,H1P60V4);
add_edge(H1P1V58f,H1P29V3);
add_edge(H1P1V59,H1P29V4);
add_edge(H1P1V60,H1P30V1);     /*link up vert edges of room 511 */
add_edge(H1P1V60a,H1P30V2);
add_edge(H1P1V60b,H1P61V1);
add_edge(H1P1V60c,H1P61V2);
add_edge(H1P1V60d,H1P61V3);
add_edge(H1P1V60e,H1P61V4);
add_edge(H1P1V60f,H1P30V3);
add_edge(H1P1V61,H1P30V4);
add_edge(H1P1V62,H1P3V4);    /*corner*/


add_edge(H1P1V63,H1P63V1);     /*link up vert edges of room elev 1*/
add_edge(H1P1V63a,H1P63V2);
add_edge(H1P1V63b,H1P31V1);
add_edge(H1P1V63c,H1P31V2);
add_edge(H1P1V63d,H1P31V3);
add_edge(H1P1V63e,H1P31V4);
add_edge(H1P1V63f,H1P31V5);
add_edge(H1P1V63g,H1P63V3);
add_edge(H1P1V64,H1P63V4);
add_edge(H1P1V65,H1P64V1);     /*link up vert edges of room elev 2*/
add_edge(H1P1V65a,H1P64V2);
add_edge(H1P1V65b,H1P32V1);
add_edge(H1P1V65c,H1P32V2);
add_edge(H1P1V65d,H1P32V3);
add_edge(H1P1V65e,H1P32V4);
add_edge(H1P1V65f,H1P32V5);
add_edge(H1P1V65g,H1P64V3);
add_edge(H1P1V66,H1P64V4);

add_edge(H1P1V67,H1P3V1);    /*corner*/
add_edge(H1P1V68,H1P33V1);     /*link up vert edges of stairwell*/
add_edge(H1P1V68a,H1P33V2);
add_edge(H1P1V68b,H1P62V1);
add_edge(H1P1V68c,H1P62V2);
add_edge(H1P1V68d,H1P62V3);
add_edge(H1P1V68e,H1P62V4);
add_edge(H1P1V68f,H1P33V3);
add_edge(H1P1V69,H1P33V4);
add_edge(H1P1V70,H1P2V4);    /*corner*/



/* Now define the different classes of doors and put instances inside the
   door jams */
```

```
add_instance("hallway",7,H1,0.0,0.0,0.0,0.0,0.0,0.0);

  H2=add_ph("office_door",11,W,0,1);
  H2P1=add_pg(H2,0.0,1,1);
  H2P1V1 = add_vertex(H2P1,0.0,0.0);
  H2P1V2 = add_vertex(H2P1,1.75,0.0);
  H2P1V3 = add_vertex(H2P1,1.75,35.5);
  H2P1V4 = add_vertex(H2P1,0.0,35.5);
  H2P2=add_pg(H2,83.5,0,1);
  H2P2V1 = add_vertex(H2P2,0.0,0.0);
  H2P2V2 = add_vertex(H2P2,1.75,0.0);
  H2P2V3 = add_vertex(H2P2,1.75,35.5);
  H2P2V4 = add_vertex(H2P2,0.0,35.5);
  add_edge(H2P1V1,H2P2V1);   /*link up vert edges of door*/
  add_edge(H2P1V2,H2P2V2);
  add_edge(H2P1V3,H2P2V3);
  add_edge(H2P1V4,H2P2V4);

  add_ceiling(H2P1,H2P2);

add_instance("door506",7,H2,-5.3,239.6,0.2,0.0,0.0,0.0);
add_instance("door510",7,H2,-5.3,713.8,0.2,0.0,0.0,0.0);
add_instance("door512",7,H2,-5.3,861.5,0.2,0.0,35.5,0.0);
add_instance("door514",7,H2,-5.3,937.6,0.2,0.0,0.0,0.0);
add_instance("door516",7,H2,-5.3,1085.3,0.2,0.0,35.5,0.0);
add_instance("door518",7,H2,-5.3,1161.8,0.2,0.0,0.0,0.0);
add_instance("door520",7,H2,-5.3,1309.0,0.2,0.0,35.5,0.0);
add_instance("door524",7,H2,-5.3,1618.9,0.2,0.0,35.5,0.0);
add_instance("door528A",8,H2,-5.3,1836.5,0.2,0.0,0.0,0.0);
add_instance("door528B",8,H2,-5.3,1919.2,0.2,0.0,0.0,0.0);
add_instance("door530A",8,H2,-5.3,2030.5,0.2,0.0,0.0,0.0);
add_instance("door530B",8,H2,-5.3,2230.7,0.2,0.0,35.5,0.0);
add_instance("door530C",8,H2,-5.3,2253.9,0.2,0.0,0.0,0.0);
add_instance("door421",7,H2,103.3,2136.3,0.2,1.75,0.0,0.0);
add_instance("door525",7,H2,103.3,1342.6,0.2,1.75,35.5,0.0);
add_instance("door523",7,H2,103.3,1118.7,0.2,1.75,35.5,0.0);
add_instance("door521",7,H2,103.3,796.0,0.2,1.75,35.5,0.0);
add_instance("door519",7,H2,103.3,564.4,0.2,1.75,35.5,0.0);
add_instance("door?",5,H2,293.8,415.65,0.2,0.0,35.5,90.0);
add_instance("door511",7,H2,342.8,310.1,0.2,1.75,35.5,0.0);
add_instance("doorstairs",10,H2,103.3,64.4,0.2,1.75,0.0,0.0);

  H3=add_ph("fire_door",9,W,0,1);
  H3P1=add_pg(H3,0.0,1,1);
  H3P1V1 = add_vertex(H3P1,0.0,0.0);
  H3P1V2 = add_vertex(H3P1,1.75,0.0);
  H3P1V3 = add_vertex(H3P1,1.75,35.6);
  H3P1V4 = add_vertex(H3P1,0.0,35.6);
  H3P2=add_pg(H3,82.9,0,1);
  H3P2V1 = add_vertex(H3P2,0.0,0.0);
```

```
  H3P2V2 = add_vertex(H3P2,1.75,0.0);
  H3P2V3 = add_vertex(H3P2,1.75,35.6);
  H3P2V4 = add_vertex(H3P2,0.0,35.6);
  add_edge(H3P1V1,H3P2V1);    /*link up vert edges of door*/
  add_edge(H3P1V2,H3P2V2);
  add_edge(H3P1V3,H3P2V3);
  add_edge(H3P1V4,H3P2V4);

  add_ceiling(H3P1,H3P2);

add_instance("1st_fire_door1",13,H3,-5.5,1488.3,0.2,0.0,0.0,0.0);
add_instance("1st_fire_door2",13,H3,-5.5,1559.7,0.2,0.0,35.6,0.0);


  H4=add_ph("restroom_door",13,W,0,1);
  H4P1=add_pg(H4,0.0,1,1);
  H4P1V1 = add_vertex(H4P1,0.0,0.0);
  H4P1V2 = add_vertex(H4P1,1.75,0.0);
  H4P1V3 = add_vertex(H4P1,1.75,31.5);
  H4P1V4 = add_vertex(H4P1,0.0,31.5);
  H4P2=add_pg(H4,83.25,0,1);
  H4P2V1 = add_vertex(H4P2,0.0,0.0);
  H4P2V2 = add_vertex(H4P2,1.75,0.0);
  H4P2V3 = add_vertex(H4P2,1.75,31.5);
  H4P2V4 = add_vertex(H4P2,0.0,31.5);
  add_edge(H4P1V1,H4P2V1);
  add_edge(H4P1V2,H4P2V2);
  add_edge(H4P1V3,H4P2V3);
  add_edge(H4P1V4,H4P2V4);

  add_ceiling(H4P1,H4P2);

add_instance("door522R",8,H4,-5.3,1461.0,0.2,0.0,31.5,0.0);
add_instance("door526R",8,H4,-5.3,1785.9,0.2,0.0,31.5,0.0);

  H5=add_ph("double_door",11,W,0,1);
  H5P1=add_pg(H5,0.0,1,1);
  H5P1V1 = add_vertex(H5P1,0.0,0.0);
  H5P1V2 = add_vertex(H5P1,1.75,0.0);
  H5P1V3 = add_vertex(H5P1,1.75,29.6);
  H5P1V4 = add_vertex(H5P1,0.0,29.6);
  H5P2=add_pg(H5,82.9,0,1);
  H5P2V1 = add_vertex(H5P2,0.0,0.0);
  H5P2V2 = add_vertex(H5P2,1.75,0.0);
  H5P2V3 = add_vertex(H5P2,1.75,29.6);
  H5P2V4 = add_vertex(H5P2,0.0,29.6);
  add_edge(H5P1V1,H5P2V1);    /*link up vert edges of door*/
  add_edge(H5P1V2,H5P2V2);
  add_edge(H5P1V3,H5P2V3);
  add_edge(H5P1V4,H5P2V4);
```

```
    add_ceiling(H5P1,H5P2);

add_instance("1door531",8,H5,103.3,1937.4,0.2,1.75,29.6,0.0);
add_instance("2door531",8,H5,103.3,1878.0,0.2,1.75,0.0,0.0);
add_instance("1door529",8,H5,103.3,1744.2,0.2,1.75,29.6,0.0);
add_instance("2door529",8,H5,103.3,1684.8,0.2,1.75,0.0,0.0);
add_instance("1door527",8,H5,103.3,1522.1,0.2,1.75,29.6,0.0);
add_instance("2door527",8,H5,103.3,1462.7,0.2,1.75,0.0,0.0);
add_instance("1door_office",12,H5,339.25,402.3,0.2,1.75,29.6,0.0);
add_instance("2door_office",12,H5,339.25,342.9,0.2,1.75,0.0,0.0);


/*   Notice that lights have no height */

  H6=add_ph("light",5,W,1,1);
  H6P1=add_pg(H6,0.0,1,1);
  H6P1V1 = add_vertex(H6P1,0.0,0.0);
  H6P1V2 = add_vertex(H6P1,45.5,0.0);
  H6P1V3 = add_vertex(H6P1,45.5,21.25);
  H6P1V4 = add_vertex(H6P1,0.0,21.25);

add_instance("light1",6,H6,26.25,98.5,102.0,0.0,0.0,0.0);
add_instance("light2",6,H6,26.25,362.75,102.0,0.0,0.0,0.0);
add_instance("light3",6,H6,26.25,651.0,102.0,0.0,0.0,0.0);
add_instance("light4",6,H6,26.25,915.25,102.0,0.0,0.0,0.0);
add_instance("light5",6,H6,26.25,1251.5,102.0,0.0,0.0,0.0);
add_instance("light6",6,H6,26.25,1539.75,102.0,0.0,0.0,0.0);
add_instance("light7",6,H6,26.25,1828.0,102.0,0.0,0.0,0.0);
add_instance("light8",6,H6,26.25,2140.25,102.0,0.0,0.0,0.0);

/* Since all molding sizes are different, we need to add a separate
   polyhedron for each one. But we still need to add one instance
   of each so it will appear in the model*/

/* 37 different molding pieces */

  H7=add_ph("molding1",8,W,1,1);
  H7P1=add_pg(H7,0.0,1,1);
  H7P1V1 = add_vertex(H7P1,0.0,0.0);
  H7P1V2 = add_vertex(H7P1,0.2,0.0);
  H7P1V3 = add_vertex(H7P1,0.2,237.5);
  H7P1V4 = add_vertex(H7P1,0.0,237.5);
  H7P2=add_pg(H7,3.875,0,1);
  H7P2V1 = add_vertex(H7P2,0.0,0.0);
  H7P2V2 = add_vertex(H7P2,0.2,0.0);
  H7P2V3 = add_vertex(H7P2,0.2,237.5);
  H7P2V4 = add_vertex(H7P2,0.0,237.5);
  add_edge(H7P1V1,H7P2V1);
  add_edge(H7P1V2,H7P2V2);
  add_edge(H7P1V3,H7P2V3);
  add_edge(H7P1V4,H7P2V4);
```

73

```
  add_ceiling(H7P1,H7P2);

add_instance("molding1",8,H7,0.0,0.0,0.0,0.0,0.0,0.0);

  H8=add_ph("molding2",8,W,1,1);
  H8P1=add_pg(H8,0.0,1,1);
  H8P1V1 = add_vertex(H8P1,0.0,0.0);
  H8P1V2 = add_vertex(H8P1,0.2,0.0);
  H8P1V3 = add_vertex(H8P1,0.2,434.5);
  H8P1V4 = add_vertex(H8P1,0.0,434.5);
  H8P2=add_pg(H8,3.875,0,1);
  H8P2V1 = add_vertex(H8P2,0.0,0.0);
  H8P2V2 = add_vertex(H8P2,0.2,0.0);
  H8P2V3 = add_vertex(H8P2,0.2,434.5);
  H8P2V4 = add_vertex(H8P2,0.0,434.5);
  add_edge(H8P1V1,H8P2V1);
  add_edge(H8P1V2,H8P2V2);
  add_edge(H8P1V3,H8P2V3);
  add_edge(H8P1V4,H8P2V4);
  add_ceiling(H8P1,H8P2);

add_instance("molding2",8,H8,0.0,277.2,0.0,0.0,0.0,0.0);

  H9=add_ph("molding3",8,W,1,1);
  H9P1=add_pg(H9,0.0,1,1);
  H9P1V1 = add_vertex(H9P1,0.0,0.0);
  H9P1V2 = add_vertex(H9P1,0.2,0.0);
  H9P1V3 = add_vertex(H9P1,0.2,72.5);
  H9P1V4 = add_vertex(H9P1,0.0,72.5);
  H9P2=add_pg(H9,3.875,0,1);
  H9P2V1 = add_vertex(H9P2,0.0,0.0);
  H9P2V2 = add_vertex(H9P2,0.2,0.0);
  H9P2V3 = add_vertex(H9P2,0.2,72.5);
  H9P2V4 = add_vertex(H9P2,0.0,72.5);
  add_edge(H9P1V1,H9P2V1);
  add_edge(H9P1V2,H9P2V2);
  add_edge(H9P1V3,H9P2V3);
  add_edge(H9P1V4,H9P2V4);
  add_ceiling(H9P1,H9P2);

add_instance("molding3",8,H9,0.0,751.4,0.0,0.0,0.0,0.0);

  H10=add_ph("molding4",8,W,1,1);
  H10P1=add_pg(H10,0.0,1,1);
  H10P1V1 = add_vertex(H10P1,0.0,0.0);
  H10P1V2 = add_vertex(H10P1,0.2,0.0);
  H10P1V3 = add_vertex(H10P1,0.2,71.9);
  H10P1V4 = add_vertex(H10P1,0.0,71.9);
  H10P2=add_pg(H10,3.875,0,1);
  H10P2V1 = add_vertex(H10P2,0.0,0.0);
  H10P2V2 = add_vertex(H10P2,0.2,0.0);
```

```
H10P2V3 = add_vertex(H10P2,0.2,71.9);
H10P2V4 = add_vertex(H10P2,0.0,71.9);
add_edge(H10P1V1,H10P2V1);
add_edge(H10P1V2,H10P2V2);
add_edge(H10P1V3,H10P2V3);
add_edge(H10P1V4,H10P2V4);
add_ceiling(H10P1,H10P2);

add_instance("molding4",8,H10,0.0,863.6,0.0,0.0,0.0,0.0);

H11=add_ph("molding5",8,W,1,1);
H11P1=add_pg(H11,0.0,1,1);
H11P1V1 = add_vertex(H11P1,0.0,0.0);
H11P1V2 = add_vertex(H11P1,0.2,0.0);
H11P1V3 = add_vertex(H11P1,0.2,72.5);
H11P1V4 = add_vertex(H11P1,0.0,72.5);
H11P2=add_pg(H11,3.875,0,1);
H11P2V1 = add_vertex(H11P2,0.0,0.0);
H11P2V2 = add_vertex(H11P2,0.2,0.0);
H11P2V3 = add_vertex(H11P2,0.2,72.5);
H11P2V4 = add_vertex(H11P2,0.0,72.5);
add_edge(H11P1V1,H11P2V1);
add_edge(H11P1V2,H11P2V2);
add_edge(H11P1V3,H11P2V3);
add_edge(H11P1V4,H11P2V4);
add_ceiling(H11P1,H11P2);

add_instance("molding5",8,H11,0.0,975.2,0.0,0.0,0.0,0.0);

H12=add_ph("molding6",8,W,1,1);
H12P1=add_pg(H12,0.0,1,1);
H12P1V1 = add_vertex(H12P1,0.0,0.0);
H12P1V2 = add_vertex(H12P1,0.2,0.0);
H12P1V3 = add_vertex(H12P1,0.2,72.3);
H12P1V4 = add_vertex(H12P1,0.0,72.3);
H12P2=add_pg(H12,3.875,0,1);
H12P2V1 = add_vertex(H12P2,0.0,0.0);
H12P2V2 = add_vertex(H12P2,0.2,0.0);
H12P2V3 = add_vertex(H12P2,0.2,72.3);
H12P2V4 = add_vertex(H12P2,0.0,72.3);
add_edge(H12P1V1,H12P2V1);
add_edge(H12P1V2,H12P2V2);
add_edge(H12P1V3,H12P2V3);
add_edge(H12P1V4,H12P2V4);
add_ceiling(H12P1,H12P2);

add_instance("molding6",8,H12,0.0,1087.4,0.0,0.0,0.0,0.0);

H13=add_ph("molding7",8,W,1,1);
H13P1=add_pg(H13,0.0,1,1);
H13P1V1 = add_vertex(H13P1,0.0,0.0);
```

```
    H13P1V2 = add_vertex(H13P1,0.2,0.0);
    H13P1V3 = add_vertex(H13P1,0.2,72.0);
    H13P1V4 = add_vertex(H13P1,0.0,72.0);
    H13P2=add_pg(H13,3.875,0,1);
    H13P2V1 = add_vertex(H13P2,0.0,0.0);
    H13P2V2 = add_vertex(H13P2,0.2,0.0);
    H13P2V3 = add_vertex(H13P2,0.2,72.0);
    H13P2V4 = add_vertex(H13P2,0.0,72.0);
    add_edge(H13P1V1,H13P2V1);
    add_edge(H13P1V2,H13P2V2);
    add_edge(H13P1V3,H13P2V3);
    add_edge(H13P1V4,H13P2V4);
    add_ceiling(H13P1,H13P2);

add_instance("molding7",8,H13,0.0,1199.4,0.0,0.0,0.0,0.0);

    H14=add_ph("molding8",8,W,1,1);
    H14P1=add_pg(H14,0.0,1,1);
    H14P1V1 = add_vertex(H14P1,0.0,0.0);
    H14P1V2 = add_vertex(H14P1,0.2,0.0);
    H14P1V3 = add_vertex(H14P1,0.2,116.5);
    H14P1V4 = add_vertex(H14P1,0.0,116.5);
    H14P2=add_pg(H14,3.875,0,1 ;
    H14P2V1 = add_vertex(H14P2,0.0,0.0);
    H14P2V2 = add_vertex(H14P2,0.2,0.0);
    H14P2V3 = add_vertex(H14P2,0.2,116.5);
    H14P2V4 = add_vertex(H14P2,0.0,116.5);
    add_edge(H14P1V1,H14P2V1);
    add_edge(H14P1V2,H14P2V2);
    add_edge(H14P1V3,H14P2V3);
    add_edge(H14P1V4,H14P2V4);
    add_ceiling(H14P1,H14P2);

add_instance("molding8",8,H14,0.0,1311.1,0.0,0.0,0.0,0.0);

    H15=add_ph("molding9",8,W,1,1);
    H15P1=add_pg(H15,0.0,1,1);
    H15P1V1 = add_vertex(H15P1,0.0,0.0);
    H15P1V2 = add_vertex(H15P1,0.2,0.0);
    H15P1V3 = add_vertex(H15P1,0.2,22.7);
    H15P1V4 = add_vertex(H15P1,0.0,22.7);
    H15P2=add_pg(H15,3.875,0,1);
    H15P2V1 = add_vertex(H15P2,0.0,0.0);
    H15P2V2 = add_vertex(H15P2,0.2,0.0);
    H15P2V3 = add_vertex(H15P2,0.2,22.7);
    H15P2V4 = add_vertex(H15P2,0.0,22.7);
    add_edge(H15P1V1,H15P2V1);
    add_edge(H15P1V2,H15P2V2);
    add_edge(H15P1V3,H15P2V3);
    add_edge(H15P1V4,H15P2V4);
    add_ceiling(H15P1,H15P2);
```

```
add_instance("molding9",8,H15,0.0,1463.3,0.0,0.0,0.0,0.0);

  H16=add_ph("molding10",9,W,1,1);
  H16P1=add_pg(H16,0.0,1,1);
  H16P1V1 = add_vertex(H16P1,0.0,0.0);
  H16P1V2 = add_vertex(H16P1,0.2,0.0);
  H16P1V3 = add_vertex(H16P1,0.2,19.3);
  H16P1V4 = add_vertex(H16P1,0.0,19.3);
  H16P2=add_pg(H16,3.875,0,1);
  H16P2V1 = add_vertex(H16P2,0.0,0.0);
  H16P2V2 = add_vertex(H16P2,0.2,0.0);
  H16P2V3 = add_vertex(H16P2,0.2,19.3);
  H16P2V4 = add_vertex(H16P2,0.0,19.3);
  add_edge(H16P1V1,H16P2V1);
  add_edge(H16P1V2,H16P2V2);
  add_edge(H16P1V3,H16P2V3);
  add_edge(H16P1V4,H16P2V4);
  add_ceiling(H16P1,H16P2);

add_instance("molding10",9,H16,0.0,1562.0,0.0,0.0,0.0,0.0);

  H17=add_ph("molding11",9,W,1,1);
  H17P1=add_pg(H17,0.0,1,1);
  H17P1V1 = add_vertex(H17P1,0.0,0.0);
  H17P1V2 = add_vertex(H17P1,0.2,0.0);
  H17P1V3 = add_vertex(H17P1,0.2,31.4);
  H17P1V4 = add_vertex(H17P1,0.0,31.4);
  H17P2=add_pg(H17,3.875,0,1);
  H17P2V1 = add_vertex(H17P2,0.0,0.0);
  H17P2V2 = add_vertex(H17P2,0.2,0.0);
  H17P2V3 = add_vertex(H17P2,0.2,31.4);
  H17P2V4 = add_vertex(H17P2,0.0,31.4);
  add_edge(H17P1V1,H17P2V1);
  add_edge(H17P1V2,H17P2V2);
  add_edge(H17P1V3,H17P2V3);
  add_edge(H17P1V4,H17P2V4);
  add_ceiling(H17P1,H17P2);

add_instance("molding11",9,H17,0.0,1619.0,0.0,0.0,0.0,0.0);

  H18=add_ph("molding12",9,W,1,1);
  H18P1=add_pg(H18,0.0,1,1);
  H18P1V1 = add_vertex(H18P1,0.0,0.0);
  H18P1V2 = add_vertex(H18P1,0.2,0.0);
  H18P1V3 = add_vertex(H18P1,0.2,68.0);
  H18P1V4 = add_vertex(H18P1,0.0,68.0);
  H18P2=add_pg(H18,3.875,0,1);
  H18P2V1 = add_vertex(H18P2,0.0,0.0);
  H18P2V2 = add_vertex(H18P2,0.2,0.0);
  H18P2V3 = add_vertex(H18P2,0.2,68.0);
```

```
    H18P2V4 = add_vertex(H18P2,0.0,68.0);
    add_edge(H18P1V1,H18P2V1);
    add_edge(H18P1V2,H18P2V2);
    add_edge(H18P1V3,H18P2V3);
    add_edge(H18P1V4,H18P2V4);
    add_ceiling(H18P1,H18P2);

add_instance("molding12",9,H18,0.0,1684.5,0.0,0.0,0.0,0.0);

    H19=add_ph("molding13",9,W,1,1);
    H19P1=add_pg(H19,0.0,1,1);
    H19P1V1 = add_vertex(H19P1,0.0,0.0);
    H19P1V2 = add_vertex(H19P1,0.2,0.0);
    H19P1V3 = add_vertex(H19P1,0.2,46.2);
    H19P1V4 = add_vertex(H19P1,0.0,46.2);
    H19P2=add_pg(H19,3.875,0,1);
    H19P2V1 = add_vertex(H19P2,0.0,0.0);
    H19P2V2 = add_vertex(H19P2,0.2,0.0);
    H19P2V3 = add_vertex(H19P2,0.2,46.2);
    H19P2V4 = add_vertex(H19P2,0.0,46.2);
    add_edge(H19P1V1,H19P2V1);
    add_edge(H19P1V2,H19P2V2);
    add_edge(H19P1V3,H19P2V3);
    add_edge(H19P1V4,H19P2V4);
    add_ceiling(H19P1,H19P2);

add_instance("molding13",9,H19,0.0,1788.2,0.0,0.0,0.0,0.0);

    H20=add_ph("molding14",9,W,1,1);
    H20P1=add_pg(H20,0.0,1,1);
    H20P1V1 = add_vertex(H20P1,0.0,0.0);
    H20P1V2 = add_vertex(H20P1,0.2,0.0);
    H20P1V3 = add_vertex(H20P1,0.2,43.0);
    H20P1V4 = add_vertex(H20P1,0.0,43.0);
    H20P2=add_pg(H20,3.875,0,1);
    H20P2V1 = add_vertex(H20P2,0.0,0.0);
    H20P2V2 = add_vertex(H20P2,0.2,0.0);
    H20P2V3 = add_vertex(H20P2,0.2,43.0);
    H20P2V4 = add_vertex(H20P2,0.0,43.0);
    add_edge(H20P1V1,H20P2V1);
    add_edge(H20P1V2,H20P2V2);
    add_edge(H20P1V3,H20P2V3);
    add_edge(H20P1V4,H20P2V4);
    add_ceiling(H20P1,H20P2);

add_instance("molding14",9,H20,0.0,1874.1,0.0,0.0,0.0,0.0);

    H21=add_ph("molding15",9,W,1,1);
    H21P1=add_pg(H21,0.0,1,1);
    H21P1V1 = add_vertex(H21P1,0.0,0.0);
    H21P1V2 = add_vertex(H21P1,0.2,0.0);
```

```
H21P1V3 = add_vertex(H21P1,0.2, 1.6);
H21P1V4 = add_vertex(H21P1,0.0,71.6);
H21P2=add_pg(H21,3.875,0,1);
H21P2V1 = add_vertex(H21P2,0.0,0.0);
H21P2V2 = add_vertex(H21P2,0.2,0.0);
H21P2V3 = add_vertex(H21P2,0.2,71.6);
H21P2V4 = add_vertex(H21P2,0.0,71.6);
add_edge(H21P1V1,H21P2V1);
add_edge(H21P1V2,H21P2V2);
add_edge(H21P1V3,H21P2V3);
add_edge(H21P1V4,H21P2V4);
add_ceiling(H21P1,H21P2);

add_instance("molding15",9,H21,0.0,1956.8,0.0,0.0,0.0,0.0);

  H22=add_ph("molding16",9,W,1,1);
  H22P1=add_pg(H22,0.0,1,1);
  H22P1V1 = add_vertex(H22P1,0.0,0.0);
  H22P1V2 = add_vertex(H22P1,0.2,0.0);
  H22P1V3 = add_vertex(H22P1,0.2,125.0);
  H22P1V4 = add_vertex(H22P1,0.0,125.0);
  H22P2=add_pg(H22,3.875,0,1);
  H22P2V1 = add_vertex(H22P2,0.0,0.0);
  H22P2V2 = add_vertex(H22P2,0.2,0.0);
  H22P2V3 = add_vertex(H22P2,0.2,125.0);
  H22P2V4 = add_vertex(H22P2,0.0,125.0);
  add_edge(H22P1V1,H22P2V1);
  add_edge(H22P1V2,H22P2V2);
  add_edge(H22P1V3,H22P2V3);
  add_edge(H22P1V4,H22P2V4);
  add_ceiling(H22P1,H22P2);

add_instance("molding16",9,H22,0.0,2068.1,0.0,0.0,0.0,0.0);

  H23=add_ph("molding9",8,W,1,1);
  H23P1=add_pg(H23,0.0,1,1);
  H23P1V1 = add_vertex(H23P1,0.0,0.0);
  H23P1V2 = add_vertex(H23P1,0.2,0.0);
  H23P1V3 = add_vertex(H23P1,0.2,19.0);
  H23P1V4 = add_vertex(H23P1,0.0,19.0);
  H23P2=add_pg(H23,3.875,0,1);
  H23P2V1 = add_vertex(H23P2,0.0,0.0);
  H23P2V2 = add_vertex(H23P2,0.2,0.0)
  H23P2V3 = add_vertex(H23P2,0.2,19.0);
  H23P2V4 = add_vertex(H23P2,0.0,19.0);
  add_edge(H23P1V1,H23P2V1);
  add_edge(H23P1V2,H23P2V2);
  add_edge(H23P1V3,H23P2V3);
  add_edge(H23P1V4,H23P2V4);
  add_ceiling(H23P1,H23P2);
```

```
add_instance("molding16",9,H23,0.0,2232.8,0.0,0.0,0.0,0.0);

  H24=add_ph("molding17",9,W,1,1);
  H24P1=add_pg(H24,0.0,1,1);
  H24P1V1 = add_vertex(H24P1,0.0,0.0);
  H24P1V2 = add_vertex(H24P1,0.2,0.0);
  H24P1V3 = add_vertex(H24P1,0.2,61.7);
  H24P1V4 = add_vertex(H24P1,0.0,61.7);
  H24P2=add_pg(H24,3.875,0,1);
  H24P2V1 = add_vertex(H24P2,0.0,0.0);
  H24P2V2 = add_vertex(H24P2,0.2,0.0);
  H24P2V3 = add_vertex(H24P2,0.2,61.7);
  H24P2V4 = add_vertex(H24P2,0.0,61.7);
  add_edge(H24P1V1,H24P2V1);
  add_edge(H24P1V2,H24P2V2);
  add_edge(H24P1V3,H24P2V3);
  add_edge(H24P1V4,H24P2V4);
  add_ceiling(H24P1,H24P2);

add_instance("molding17",9,H24,0.0,2289.5,0.0,0.0,0.0,0.0);

  H25=add_ph("molding18",9,W,1,1);
  H25P1=add_pg(H25,0.0,1,1);
  H25P1V1 = add_vertex(H25P1,0.0,0.0);
  H25P1V2 = add_vertex(H25P1,0.0,177.3);
  H25P1V3 = add_vertex(H25P1,-0.2,177.3);
  H25P1V4 = add_vertex(H25P1,-0.2,0.0);
  H25P2=add_pg(H25,3.875,0,1);
  H25P2V1 = add_vertex(H25P2,0.0,0.0);
  H25P2V2 = add_vertex(H25P2,0.0,177.3);
  H25P2V3 = add_vertex(H25P2,-0.2,177.3);
  H25P2V4 = add_vertex(H25P2,-0.2,0.0);
  add_edge(H25P1V1,H25P2V1);
  add_edge(H25P1V2,H25P2V2);
  add_edge(H25P1V3,H25P2V3);
  add_edge(H25P1V4,H25P2V4);
  add_ceiling(H25P1,H25P2);

add_instance("molding18",9,H25,98.0,2173.9,0.0,0.0,0.0,0.0);

  H26=add_ph("molding19",9,W,1,1);
  H26P1=add_pg(H26,0.0,1,1);
  H26P1V1 = add_vertex(H26P1,0.0,0.0);
  H26P1V2 = add_vertex(H26P1,0.0,194.5);
  H26P1V3 = add_vertex(H26P1,-0.2,194.5);
  H26P1V4 = add_vertex(H26P1,-0.2,0.0);
  H26P2=add_pg(H26,3.875,0,1);
  H26P2V1 = add_vertex(H26P2,0.0,0.0);
  H26P2V2 = add_vertex(H26P2,0.0,194.5);
  H26P2V3 = add_vertex(H26P2,-0.2,194.5);
  H26P2V4 = add_vertex(H26P2,-0.2,0.0);
```

```
    add_edge(H26P1V1,H26P2V1);
    add_edge(H26P1V2,H26P2V2);
    add_edge(H26P1V3,H26P2V3);
    add_edge(H26P1V4,H26P2V4);
    add_ceiling(H26P1,H26P2);

add_instance("molding19",9,H26,98.0,1939.7,0.0,0.0,0.0,0.0);


    H27=add_ph("molding20",9,W,1,1);
    H27P1=add_pg(H27,0.0,1,1);
    H27P1V1 = add_vertex(H27P1,0.0,0.0);
    H27P1V2 = add_vertex(H27P1,0.0,129.2);
    H27P1V3 = add_vertex(H27P1,-0.2,129.2);
    H27P1V4 = add_vertex(H27P1,-0.2,0.0);
    H27P2=add_pg(H27,3.875,0,1);
    H27P2V1 = add_vertex(H27P2,0.0,0.0);
    H27P2V2 = add_vertex(H27P2,0.0,129.2);
    H27P2V3 = add_vertex(H27P2,-0.2,129.2);
    H27P2V4 = add_vertex(H27P2,-0.2,0.0);
    add_edge(H27P1V1,H27P2V1);
    add_edge(H27P1V2,H27P2V2);
    add_edge(H27P1V3,H27P2V3);
    add_edge(H27P1V4,H27P2V4);
    add_ceiling(H27P1,H27P2);

add_instance("molding20",9,H27,98.0,1746.5,0.0,0.0,0.0,0.0);

    H28=add_ph("molding21",9,W,1,1);
    H28P1=add_pg(H28,0.0,1,1);
    H28P1V1 = add_vertex(H28P1,0.0,0.0);
    H28P1V2 = add_vertex(H28P1,0.0,158.1);
    H28P1V3 = add_vertex(H28P1,-0.2,158.1);
    H28P1V4 = add_vertex(H28P1,-0.2,0.0);
    H28P2=add_pg(H28,3.875,0,1);
    H28P2V1 = add_vertex(H28P2,0.0,0.0);
    H28P2V2 = add_vertex(H28P2,0.0,158.1);
    H28P2V3 = add_vertex(H28P2,-0.2,158.1);
    H28P2V4 = add_vertex(H28P2,-0.2,0.0);
    add_edge(H28P1V1,H28P2V1);
    add_edge(H28P1V2,H28P2V2);
    add_edge(H28P1V3,H28P2V3);
    add_edge(H28P1V4,H28P2V4);
    add_ceiling(H28P1,H28P2);

add_instance("molding21",9,H28,98.0,1524.4,0.0,0.0,0.0,0.0);

    H29=add_ph("molding22",9,W,1,1);
    H29P1=add_pg(H29,0.0,1,1);
    H29P1V1 = add_vertex(H29P1,0.0,0.0);
    H29P1V2 = add_vertex(H29P1,0.0,115.7);
```

```
    H29P1V3 = add_vertex(H29P1,-0.2,115.7);
    H29P1V4 = add_vertex(H29P1,-0.2,0.0);
    H29P2=add_pg(H29,3.875,0,1);
    H29P2V1 = add_vertex(H29P2,0.0,0.0);
    H29P2V2 = add_vertex(H29P2,0.0,115.7);
    H29P2V3 = add_vertex(H29P2,-0.2,115.7);
    H29P2V4 = add_vertex(H29P2,-0.2,0.0);
    add_edge(H29P1V1,H29P2V1);
    add_edge(H29P1V2,H29P2V2);
    add_edge(H29P1V3,H29P2V3);
    add_edge(H29P1V4,H29P2V4);
    add_ceiling(H29P1,H29P2);

add_instance("molding22",9,H29,98.0,1344.7,0.0,0.0,0.0,0.0);

    H30=add_ph("molding23",9,W,1,1);
    H30P1=add_pg(H30,0.0,1,1);
    H30P1V1 = add_vertex(H30P1,0.0,0.0);
    H30P1V2 = add_vertex(H30P1,0.0,184.2);
    H30P1V3 = add_vertex(H30P1,-0.2,184.2);
    H30P1V4 = add_vertex(H30P1,-0.2,0.0);
    H30P2=add_pg(H30,3.875,0,1);
    H30P2V1 = add_vertex(H30P2,0.0,0.0);
    H30P2V2 = add_vertex(H30P2,0.0,184.2);
    H30P2V3 = add_vertex(H30P2,-0.2,184.2);
    H30P2V4 = add_vertex(H30P2,-0.2,0.0);
    add_edge(H30P1V1,H30P2V1);
    add_edge(H30P1V2,H30P2V2);
    add_edge(H30P1V3,H30P2V3);
    add_edge(H30P1V4,H30P2V4);
    add_ceiling(H30P1,H30P2);

add_instance("molding23",9,H30,98.0,1120.8,0.0,0.0,0.0,0.0);

    H46=add_ph("molding24",9,W,1,1);
    H46P1=add_pg(H46,0.0,1,1);
    H46P1V1 = add_vertex(H46P1,0.0,0.0);
    H46P1V2 = add_vertex(H46P1,0.0,202.0);
    H46P1V3 = add_vertex(H46P1,-0.2,202.0);
    H46P1V4 = add_vertex(H46P1,-0.2,0.0);
    H46P2=add_pg(H46,3.875,0,1);
    H46P2V1 = add_vertex(H46P2,0.0,0.0);
    H46P2V2 = add_vertex(H46P2,0.0,202.0);
    H46P2V3 = add_vertex(H46P2,-0.2,202.0);
    H46P2V4 = add_vertex(H46P2,-0.2,0.0);
    add_edge(H46P1V1,H46P2V1);
    add_edge(H46P1V2,H46P2V2);
    add_edge(H46P1V3,H46P2V3);
    add_edge(H46P1V4,H46P2V4);
    add_ceiling(H46P1,H46P2);
```

```
add_instance("molding24",9,H46,98.0,877.1,0.0,0.0,0.0,0.0);


  H31=add_ph("molding25",9,W,1,1);
  H31P1=add_pg(H31,0.0,1,1);
  H31P1V1 = add_vertex(H31P1,0.0,0.0);
  H31P1V2 = add_vertex(H31P1,0.0,32.0);
  H31P1V3 = add_vertex(H31P1,-0.2,32.0);
  H31P1V4 = add_vertex(H31P1,-0.2,0.0);
  H31P2=add_pg(H31,3.875,0,1);
  H31P2V1 = add_vertex(H31P2,0.0,0.0);
  H31P2V2 = add_vertex(H31P2,0.0,32.0);
  H31P2V3 = add_vertex(H31P2,-0.2,32.0);
  H31P2V4 = add_vertex(H31P2,-0.2,0.0);
  add_edge(H31P1V1,H31P2V1);
  add_edge(H31P1V2,H31P2V2);
  add_edge(H31P1V3,H31P2V3);
  add_edge(H31P1V4,H31P2V4);
  add_ceiling(H31P1,H31P2);

add_instance("molding25",9,H31,98.0,798.1,0.0,0.0,0.0,0.0);

  H32=add_ph("molding26",9,W,1,1);
  H32P1=add_pg(H32,0.0,1,1);
  H32P1V1 = add_vertex(H32P1,0.0,0.0);
  H32P1V2 = add_vertex(H32P1,0.0,191.9);
  H32P1V3 = add_vertex(H32P1,-0.2,191.9);
  H32P1V4 = add_vertex(H32P1,-0.2,0.0);
  H32P2=add_pg(H32,3.875,0,1);
  H32P2V1 = add_vertex(H32P2,0.0,0.0);
  H32P2V2 = add_vertex(H32P2,0.0,191.9);
  H32P2V3 = add_vertex(H32P2,-0.2,191.9);
  H32P2V4 = add_vertex(H32P2,-0.2,0.0);
  add_edge(H32P1V1,H32P2V1);
  add_edge(H32P1V2,H32P2V2);
  add_edge(H32P1V3,H32P2V3);
  add_edge(H32P1V4,H32P2V4);
  add_ceiling(H32P1,H32P2);

add_instance("molding26",9,H32,98.0,566.5,0.0,0.0,0.0,0.0);

  H33=add_ph("molding27",9,W,1,1);
  H33P1=add_pg(H33,0.0,1,1);
  H33P1V1 = add_vertex(H33P1,0.0,0.0);
  H33P1V2 = add_vertex(H33P1,0.0,112.9);
  H33P1V3 = add_vertex(H33P1,-0.2,112.9);
  H33P1V4 = add_vertex(H33P1,-0.2,0.0);
  H33P2=add_pg(H33,3.875,0,1);
  H33P2V1 = add_vertex(H33P2,0.0,0.0);
  H33P2V2 = add_vertex(H33P2,0.0,112.9);
  H33P2V3 = add_vertex(H33P2,-0.2,112.9);
```

```
    H33P2V4 = add_vertex(H33P2,-0.2,0.0);
    add_edge(H33P1V1,H33P2V1);
    add_edge(H33P1V2,H33P2V2);
    add_edge(H33P1V3,H33P2V3);
    add_edge(H33P1V4,H33P2V4);
    add_ceiling(H33P1,H33P2);

add_instance("molding27",9,H33,98.0,413.9,0.0,0.0,0.0,0.0);

    H34=add_ph("molding28",9,W,1,1);
    H34P1=add_pg(H34,0.0,1,1);
    H34P1V1 = add_vertex(H34P1,0.0,0.0);
    H34P1V2 = add_vertex(H34P1,0.0,-0.2);
    H34P1V3 = add_vertex(H34P1,157.9,-0.2);
    H34P1V4 = add_vertex(H34P1,157.9,0.0);
    H34P2=add_pg(H34,3.875,0,1);
    H34P2V1 = add_vertex(H34P2,0.0,0.0);
    H34P2V2 = add_vertex(H34P2,0.0,-0.2);
    H34P2V3 = add_vertex(H34P2,157.9,-0.2);
    H34P2V4 = add_vertex(H34P2,157.9,0.0);
    add_edge(H34P1V1,H34P2V1);
    add_edge(H34P1V2,H34P2V2);
    add_edge(H34P1V3,H34P2V3);
    add_edge(H34P1V4,H34P2V4);
    add_ceiling(H34P1,H34P2);

add_instance("molding28",9,H34,98.0,413.9,0.0,0.0,0.0,0.0);

    H35=add_ph("molding29",9,W,1,1);
    H35P1=add_pg(H35,0.0,1,1);
    H35P1V1 = add_vertex(H35P1,0.0,0.0);
    H35P1V2 = add_vertex(H35P1,0.0,-0.2);
    H35P1V3 = add_vertex(H35P1,41.6,-0.2);
    H35P1V4 = add_vertex(H35P1,41.6,0.0);
    H35P2=add_pg(H35,3.875,0,1);
    H35P2V1 = add_vertex(H35P2,0.0,0.0);
    H35P2V2 = add_vertex(H35P2,0.0,-0.2);
    H35P2V3 = add_vertex(H35P2,41.6,-0.2);
    H35P2V4 = add_vertex(H35P2,41.6,0.0);
    add_edge(H35P1V1,H35P2V1);
    add_edge(H35P1V2,H35P2V2);
    add_edge(H35P1V3,H35P2V3);
    add_edge(H35P1V4,H35P2V4);
    add_ceiling(H35P1,H35P2);

add_instance("molding29",9,H35,295.9,413.9,0.0,0.0,0.0,0.0);

    H36=add_ph("molding30",9,W,1,1);
    H36P1=add_pg(H36,0.0,1,1);
    H36P1V1 = add_vertex(H36P1,0.0,0.0);
    H36P1V2 = add_vertex(H36P1,-0.2,0.0);
```

```
H36P1V3 = add_vertex(H36P1,-0.2,9.3);
H36P1V4 = add_vertex(H36P1,0.0,9.3);
H36P2=add_pg(H36,3.875,0,1);
H36P2V1 = add_vertex(H36P2,0.0,0.0);
H36P2V2 = add_vertex(H36P2,-0.2,0.0);
H36P2V3 = add_vertex(H36P2,-0.2,9.3);
H36P2V4 = add_vertex(H36P2,0.0,9.3);
add_edge(H36P1V1,H36P2V1);
add_edge(H36P1V2,H36P2V2);
add_edge(H36P1V3,H36P2V3);
add_edge(H36P1V4,H36P2V4);
add_ceiling(H36P1,H36P2);

add_instance("molding30",9,H36,337.5,404.6,0.0,0.0,0.0,0.0);

H37=add_ph("molding31",9,W,1,1);
H37P1=add_pg(H37,0.0,1,1);
H37P1V1 = add_vertex(H37P1,0.0,0.0);
H37P1V2 = add_vertex(H37P1,-0.2,0.0);
H37P1V3 = add_vertex(H37P1,-0.2,28.4);
H37P1V4 = add_vertex(H37P1,0.0,28.4);
H37P2=add_pg(H37,3.875,0,1);
H37P2V1 = add_vertex(H37P2,0.0,0.0);
H37P2V2 = add_vertex(H37P2,-0.2,0.0);
H37P2V3 = add_vertex(H37P2,-0.2,28.4);
H37P2V4 = add_vertex(H37P2,0.0,28.4);
add_edge(H37P1V1,H37P2V1);
add_edge(H37P1V2,H37P2V2);
add_edge(H37P1V3,H37P2V3);
add_edge(H37P1V4,H37P2V4);
add_ceiling(H37P1,H37P2);

add_instance("molding31",9,H37,337.5,312.2,0.0,0.0,0.0,0.0);

H38=add_ph("molding32",9,W,1,1);
H38P1=add_pg(H38,0.0,1,1);
H38P1V1 = add_vertex(H38P1,0.0,0.0);
H38P1V2 = add_vertex(H38P1,-0.2,0.0);
H38P1V3 = add_vertex(H38P1,-0.2,5.1);
H38P1V4 = add_vertex(H38P1,0.0,5.1);
H38P2=add_pg(H38,3.875,0,1);
H38P2V1 = add_vertex(H38P2,0.0,0.0);
H38P2V2 = add_vertex(H38P2,-0.2,0.0);
H38P2V3 = add_vertex(H38P2,-0.2,5.1);
H38P2V4 = add_vertex(H38P2,0.0,5.1);
add_edge(H38P1V1,H38P2V1);
add_edge(H38P1V2,H38P2V2);
add_edge(H38P1V3,H38P2V3);
add_edge(H38P1V4,H38P2V4);
add_ceiling(H38P1,H38P2);
```

```
add_instance("molding32",9,H38,337.5,267.4,0.0,0.0,0.0,0.0);

   H39=add_ph("molding33",9,W,1,1);
   H39P1=add_pg(H39,0.0,1,1);
   H39P1V1 = add_vertex(H39P1,0.0,0.0);
   H39P1V2 = add_vertex(H39P1,30.6,0.0);
   H39P1V3 = add_vertex(H39P1,30.6,0.2);
   H39P1V4 = add_vertex(H39P1,0.0,0.2);
   H39P2=add_pg(H39,3.875,0,1);
   H39P2V1 = add_vertex(H39P2,0.0,0.0);
   H39P2V2 = add_vertex(H39P2,30.6,0.0);
   H39P2V3 = add_vertex(H39P2,30.6,0.2);
   H39P2V4 = add_vertex(H39P2,0.0,0.2);
   add_edge(H39P1V1,H39P2V1);
   add_edge(H39P1V2,H39P2V2);
   add_edge(H39P1V3,H39P2V3);
   add_edge(H39P1V4,H39P2V4);
   add_ceiling(H39P1,H39P2);

add_instance("molding33",9,H39,306.9,267.4,0.0,0.0,0.0,0.0);

   H40=add_ph("molding34",9,W,1,1);
   H40P1=add_pg(H40,0.0,1,1);
   H40P1V1 = add_vertex(H40P1,0.0,0.0);
   H40P1V2 = add_vertex(H40P1,56.7,0.0);
   H40P1V3 = add_vertex(H40P1,56.7,0.2);
   H40P1V4 = add_vertex(H40P1,0.0,0.2);
   H40P2=add_pg(H40,3.875,0,1);
   H40P2V1 = add_vertex(H40P2,0.0,0.0);
   H40P2V2 = add_vertex(H40P2,56.7,0.0);
   H40P2V3 = add_vertex(H40P2,56.7,0.2);
   H40P2V4 = add_vertex(H40P2,0.0,0.2);
   add_edge(H40P1V1,H40P2V1);
   add_edge(H40P1V2,H40P2V2);
   add_edge(H40P1V3,H40P2V3);
   add_edge(H40P1V4,H40P2V4);
   add_ceiling(H40P1,H40P2);

add_instance("molding34",9,H40,192.2,267.4,0.0,0.0,0.0,0.0);

   H41=add_ph("molding35",9,W,1,1);
   H41P1=add_pg(H41,0.0,1,1);
   H41P1V1 = add_vertex(H41P1,0.0,0.0);
   H41P1V2 = add_vertex(H41P1,36.2,0.0);
   H41P1V3 = add_vertex(H41P1,36.2,0.2);
   H41P1V4 = add_vertex(H41P1,0.0,0.2);
   H41P2=add_pg(H41,3.875,0,1);
   H41P2V1 = add_vertex(H41P2,0.0,0.0);
   H41P2V2 = add_vertex(H41P2,36.2,0.0);
   H41P2V3 = add_vertex(H41P2,36.2,0.2);
   H41P2V4 = add_vertex(H41P2,0.0,0.2);
```

```
    add_edge(H41P1V1,H41P2V1);
    add_edge(H41P1V2,H41P2V2);
    add_edge(H41P1V3,H41P2V3);
    add_edge(H41P1V4,H41P2V4);
    add_ceiling(H41P1,H41P2);

add_instance("molding35",9,H41,98.0,267.4,0.0,0.0,0.0,0.0);

    H42=add_ph("molding36",9,W,1,1);
    H42P1=add_pg(H42,0.0,1,1);
    H42P1V1 = add_vertex(H42P1,0.0,0.0);
    H42P1V2 = add_vertex(H42P1,0.0,-0.2);
    H42P1V3 = add_vertex(H42P1,165.4,-0.2);
    H42P1V4 = add_vertex(H42P1,165.4,0.0);
    H42P2=add_pg(H42,3.875,0,1);
    H42P2V1 = add_vertex(H42P2,0.0,0.0);
    H42P2V2 = add_vertex(H42P2,0.0,-0.2);
    H42P2V3 = add_vertex(H42P2,165.4,-0.2);
    H42P2V4 = add_vertex(H42P2,165.4,0.0);
    add_edge(H42P1V1,H42P2V1);
    add_edge(H42P1V2,H42P2V2);
    add_edge(H42P1V3,H42P2V3);
    add_edge(H42P1V4,H42P2V4);
    add_ceiling(H42P1,H42P2);

add_instance("molding36",9,H42,98.0,102.0,0.0,0.0,0.0,0.0);

    H43=add_ph("molding37",9,W,1,1);
    H43P1=add_pg(H43,0.0,1,1);
    H43P1V1 = add_vertex(H43P1,0.0,0.0);
    H43P1V2 = add_vertex(H43P1,-0.2,0.0);
    H43P1V3 = add_vertex(H43P1,-0.2,62.3);
    H43P1V4 = add_vertex(H43P1,0.0,62.3);
    H43P2=add_pg(H43,3.875,0,1);
    H43P2V1 = add_vertex(H43P2,0.0,0.0);
    H43P2V2 = add_vertex(H43P2,-0.2,0.0);
    H43P2V3 = add_vertex(H43P2,-0.2,62.3);
    H43P2V4 = add_vertex(H43P2,0.0,62.3);
    add_edge(H43P1V1,H43P2V1);
    add_edge(H43P1V2,H43P2V2);
    add_edge(H43P1V3,H43P2V3);
    add_edge(H43P1V4,H43P2V4);
    add_ceiling(H43P1,H43P2);

add_instance("molding37",9,H43,98.0,0.0,0.0,0.0,0.0,0.0);

    H44=add_ph("electric_panel",14,W,1,1);
    H44P1=add_pg(H44,0.0,1,1);
    H44P1V1=add_vertex(H44P1,0.0,0.0);
    H44P1V2=add_vertex(H44P1,0.0,47.0);
    H44P1V3=add_vertex(H44P1,-0.2,47.0);
```

```
H44P1V4=add_vertex(H44P1,-0.2,0.0);
H44P2=add_pg(H44,86.0,0,1);
H44P2V1=add_vertex(H44P2,0.0,0.0);
H44P2V2=add_vertex(H44P2,0.0,47.0);
H44P2V3=add_vertex(H44P2,-0.2,47.0);
H44P2V4=add_vertex(H44P2,-0.2,0.0);
add_edge(H44P1V1,H44P2V1);
add_edge(H44P1V2,H44P2V2);
add_edge(H44P1V3,H44P2V3);
add_edge(H44P1V4,H44P2V4);
add_ceiling(H44P1,H44P2);

add_instance("electric_panel1",15,H44,98.0,830.1,0.0,0.0,0.0,0.0);

H45=add_ph("room_label",10,W,1,1);
H45P1=add_pg(H45,57.6,1,1);
H45P1V1=add_vertex(H45P1,0.0,0.0);
H45P1V2=add_vertex(H45P1,0.0,7.0);
H45P1V3=add_vertex(H45P1,-0.2,7.0);
H45P1V4=add_vertex(H45P1,-0.2,0.0);
H45P2=add_pg(H45,64.6,0,1);
H45P2V1=add_vertex(H45P2,0.0,0.0);
H45P2V2=add_vertex(H45P2,0.0,7.0);
H45P2V3=add_vertex(H45P2,-0.2,7.0);
H45P2V4=add_vertex(H45P2,-0.2,0.0);
add_edge(H45P1V1,H45P2V1);
add_edge(H45P1V2,H45P2V2);
add_edge(H45P1V3,H45P2V3);
add_edge(H45P1V4,H45P2V4);
add_ceiling(H45P1,H45P2);

add_instance("office_label",12,H45,337.5,331.1,0.0,0.0,0.0,0.0);
add_instance("511_label",9,H45,337.5,314.9,0.0,0.0,0.0,0.0);

H47=add_ph("elevator_control",16,W,1,1);
H47P1=add_pg(H47,42.0,1,1);
H47P1V1=add_vertex(H47P1,0.0,0.0);
H47P1V2=add_vertex(H47P1,7.0,0.0);
H47P1V3=add_vertex(H47P1,7.0,0.2);
H47P1V4=add_vertex(H47P1,0.0,0.2);
H47P2=add_pg(H47,64.6,0,1);
H47P2V1=add_vertex(H47P2,0.0,0.0);
H47P2V2=add_vertex(H47P2,7.0,0.0);
H47P2V3=add_vertex(H47P2,7.0,0.2);
H47P2V4=add_vertex(H47P2,0.0,0.2);
add_edge(H47P1V1,H47P2V1);
add_edge(H47P1V2,H47P2V2);
add_edge(H47P1V3,H47P2V3);
add_edge(H47P1V4,H47P2V4);
add_ceiling(H47P1,H47P2);
```

```
add_instance("elevator_control",16,H47,216.9,267.4,0.0,0.0,0.0,0.0);

  H48=add_ph("bulletin_board",14,W,1,1);
  H48P1=add_pg(H48,36.5,1,1);
  H48P1V1=add_vertex(H48P1,0.0,0.0);
  H48P1V2=add_vertex(H48P1,0.0,-3.25);
  H48P1V3=add_vertex(H48P1,144.1,-3.25);
  H48P1V4=add_vertex(H48P1,144.1,0.0);
  H48P2=add_pg(H48,84.5,0,1);
  H48P2V1=add_vertex(H48P2,0.0,0.0);
  H48P2V2=add_vertex(H48P2,0.0,-3.25);
  H48P2V3=add_vertex(H48P2,144.1,-3.25);
  H48P2V4=add_vertex(H48P2,144.1,0.0);
  add_edge(H48P1V1,H48P2V1);
  add_edge(H48P1V2,H48P2V2);
  add_edge(H48P1V3,H48P2V3);
  add_edge(H48P1V4,H48P2V4);
  add_ceiling(H48P1,H48P2);

add_instance("bulletin_board",14,H48,105.9,413.9,0.0,0.0,0.0,0.0);

  return W;          /*return pointer to this entire world structure*/
}
```

```
/*

            Lt James Stein


This file contains the routines neccessary to support the projection of
our
2d+ model world into a 2 dimensional window. This view will then be used
for
pattern matching against the processed images extracted from the raw
camera
data.
*/

#define CCD (2.0/3.0)
#define VIEW_ANGLE 300.0
#define NEARCLIP 1.24
#define FARCLIP 5000.0
#define MAX_X 646.0
#define MAX_Y 587.28        /* changed fm 587.28 on 29 Jun 93 */


typedef struct line (
            double X1,X2,Y1,Y2,Z1,Z2;
            double MODEL_X, MODEL_Y;
            int CLIP1[6],CLIP2[6];
            struct line *NEXT;

            /* pattern-matching parameters */
            char   name[2];
            float length;
            double est_pose_orient, est_angle_to_image_center;
            int    sum_img_lines;
            float sum_conf, sum_dist, sum_scale;
) LINE;


typedef struct line_head (
            int LINES,VERT_LINES;
            LINE *LINE_LIST,*VLINE_LIST, *TAIL,*VTAIL;
) LINE_HEAD;


typedef struct window (
            double XMIN, XMAX, YMIN, YMAX,
                ZMIN, ZMAX;
) WINDOW;
```

```
/*                               Y
                                 0


         -X 90                              -90      X



                                 180
                                 -Y


NOTE: convert angles and tenths degrees to rads for sin and cos functions

*/



LINE_HEAD *conduct_visibility_sweep(WORLD*,double,double,double);



float find_z(PH,V)
     POLYHEDRON *PH;
     VERTEX *V;
{
  POLYGON *NEXT_PG;
  VERTEX  *NEXT_V;
  float Z_VALUE=66.6;
  int FOUND=0,PG_CNT=0;


  NEXT_PG=PH->POLYGON_LIST;
  while (NEXT_PG) {
    PG_CNT++;
    NEXT_V=NEXT_PG->VERTEX_LIST;
    while (NEXT_V) {
      if (NEXT_V==V) {
            Z_VALUE=NEXT_PG->Z_VALUE;
            NEXT_V=NULL;
        FOUND=1;
      }
      else {
            NEXT_V=NEXT_V->NEXT;
      }
    } /* end while */
    if (FOUND==0)
      NEXT_PG=NEXT_PG->NEXT;
    else
            NEXT_PG=NULL;
```

```c
        } /* end while */
    return (Z_VALUE);
} /* end find_z */




WINDOW *calc_window(X,Y,Z,ORIENT,FOCAL_LEN)

            double X,Y,Z,ORIENT,FOCAL_LEN;
{
            WINDOW *WIN;
            double  HYP,XMIN,YMIN,ZMIN, RADS;


  WIN=(WINDOW *)malloc(sizeof(WINDOW));

  RADS=VIEW_ANGLE/10.0*PI/180.0;
/*printf("\n\nView angle= %.21f (%.21f rads)",VIEW_ANGLE,RADS);
printf("\n\nFocal length= %.21f",FOCAL_LEN);*/
            HYP = FOCAL_LEN/cos(RADS/2.0);
/*printf("\n\nHyp len= %.21f",HYP);*/
            WIN->XMIN = X-cos((90.0-ORIENT-VIEW_ANGLE/20.0)*PI/180.0) *
HYP;
            WIN->XMAX = X-sin((ORIENT-VIEW_ANGLE/20.0)*PI/180.0) * HYP;
            WIN->YMIN = Y+sin((90.0-ORIENT-VIEW_ANGLE/20.0)*PI/180.0) *
HYP;
            WIN->YMAX = Y+cos((ORIENT-VIEW_ANGLE/20.0)*PI/180.0) * HYP;
            WIN->ZMIN = Z-CCD/2.0;
            WIN->ZMAX = Z+CCD/2.0;
            return WIN;
} /* end calc_window */




void lprint_l(L)
            LINE *L;
{
            printf("\n\nline: X1=%.21f Y1=%.21f Z1=%.21f ",L->X1,L->Y1,L-
>Z1);
            printf("\n      X2=%.21f Y2=%.21f Z2=%.21f \n",L->X2,L->Y2,L-
>Z2);
fflush(stdout);
}
```

```c
void lprint_llist(LIST)
            LINE_HEAD *LIST;
{
            LINE *NEXT_L=LIST->VLINE_LIST;

            printf("\n\n\nVertical lines (%d) are:\n\n",LIST-
>VERT_LINES);
            while (NEXT_L) {
                        lprint_l(NEXT_L);
                        NEXT_L=NEXT_L->NEXT;
            }
            printf("\n\n\nnon-vertical lines (%d) are:\n\n",LIST-
>LINES);
            fflush(stdout);
            NEXT_L=LIST->LINE_LIST;
            while (NEXT_L) {
                        lprint_l(NEXT_L);
                        NEXT_L=NEXT_L->NEXT;
            }
}




void print_line(L)
            LINE *L;
{
            printf("\nX1:%.4lf Y1:%.4lf Z1:%.4lf X2:%.4lf Y2:%.4lf
Z2:%.4lf ",
                        L->X1,L->Y1,L->Z1,L->X2,L->Y2,L->Z2);
}   /* end print_line */




LINE *make_line(I,V1,V2,Z1,Z2)
            INSTANCE *I;
            VERTEX *V1, *V2;
            double   Z1,Z2;
{
            LINE *NEW_LINE;
            double LOCAL_X,LOCAL_Y, ROT_X,ROT_Y, RADS;


  NEW_LINE = (LINE *)malloc(sizeof(LINE));
  NEW_LINE->NEXT = NULL;

  /* adjust all local coordinates to pivot point*/
```

93

```c
    LOCAL_X = V1->X - I->PIVOT_X;
    LOCAL_Y = V1->Y - I->PIVOT_Y;


    /* rotate about the z axis */
    RADS = I->ROTATION * PI / 180.0 ;    /* convert degs to rads */
    ROT_X = (cos(RADS)*LOCAL_X)+(sin(RADS)*LOCAL_Y);
    ROT_Y = (cos(RADS)*LOCAL_Y)-(sin(RADS)*LOCAL_X);


    /* translate to proper position in world model */


    NEW_LINE->X1 = I->X + ROT_X;
    NEW_LINE->Y1 = I->Y + ROT_Y;
    NEW_LINE->Z1 = I->Z + Z1;


    /* calc second vertex */
    LOCAL_X = V2->X - I->PIVOT_X;
    LOCAL_Y = V2->Y - I->PIVOT_Y;


    /* rotate about the z axis */
    RADS = I->ROTATION * PI / 180.0 ;    /* convert degs to rads */
    ROT_X = (cos(RADS)*LOCAL_X)+(sin(RADS)*LOCAL_Y);
    ROT_Y = (cos(RADS)*LOCAL_Y)-(sin(RADS)*LOCAL_X);


    /* translate to proper position in world model */
    NEW_LINE->X2 = I->X + ROT_X;
    NEW_LINE->Y2 = I->Y + ROT_Y;
    NEW_LINE->Z2 = I->Z + Z2;


    return NEW_LINE;
}    /* end make_line */



void add_lines(LIST,L)

            LINE_HEAD  *LIST;
            LINE       *L;
{

            LINE *NEXT_LINE;

            if (LIST->LINE_LIST==NULL) {
                LIST->LINE_LIST=L;
                LIST->TAIL=L;
                LIST->LINES=1;
            }
            else {
                LIST->TAIL->NEXT=L;
                LIST->LINES++;
                LIST->TAIL=L;
            }
}    /* end add_lines */
```

```
LINE_HEAD *create_line_head()
{
            LINE_HEAD *LH;

            if ((LH=(LINE_HEAD *)malloc(sizeof(LINE_HEAD)))==NULL)
                        printf("\n\ncannot create line head\n");
            LH->LINES = 0;
            LH->LINE_LIST = NULL;
            LH->TAIL = NULL;

            return LH;
}    /* end create_line_head */




void print_line_list(LH)
            LINE_HEAD *LH;
{
            LINE *NEXT_L;

            NEXT_L=LH->LINE_LIST;
            printf("\n\nThere are %d lines: \n\n",LH->LINES);
            while (NEXT_L) {
                        print_line(NEXT_L);
                        NEXT_L=NEXT_L->NEXT;
            }
}




void free_lines(LH)
            LINE_HEAD *LH;
{
            LINE *NEXT_L, *TRASH;

            NEXT_L=LH->LINE_LIST;
            while (NEXT_L) {
                        TRASH=NEXT_L;
                        NEXT_L=NEXT_L->NEXT;
                        free(TRASH);
            }
            NEXT_L=LH->VLINE_LIST;
            while (NEXT_L) {
                        TRASH=NEXT_L;
                        NEXT_L=NEXT_L->NEXT;
                        free(TRASH);
```

```
                )
                free(LH);
)    /* end free_lines */




void scale_line(L,SX,SY,SZ)

           LINE *L;
           double SX,SY,SZ;
(
           L->X1 = L->X1 * SX ;
           L->X2 = L->X2 * SX ;
           L->Y1 = L->Y1 * SY ;
           L->Y2 = L->Y2 * SY ;
           L->Z1 = L->Z1 * SZ ;
           L->Z2 = L->Z2 * SZ ;
)    /* end scale_line */




void scale_window(W,SX,SY,SZ)

           WINDOW *W;
           double SX,SY,SZ;
(
           W->XMIN = W->XMIN * SX ;
           W->XMAX = W->XMAX * SX ;
           W->YMIN = W->YMIN * SY ;
           W->YMAX = W->YMAX * SY ;
           W->ZMIN = W->ZMIN * SZ ;
           W->ZMAX = W->ZMAX * SZ ;
)    /* end scale_line */




/* shift from world coordinates to machine coordinates */
void shift_coord_line(L)
     LINE *L;
(
  double TEMP1, TEMP2;

  TEMP1 = L->Y1;
  TEMP2 = L->Y2;
  L->Y1 = L->Z1;
  L->Y2 = L->Z2;
  L->Z1 = TEMP1;
```

```c
    L->Z2 = TEMP2;
} /* end shift_coord_line */



/* shift from world coordinates to machine coordinates */
void shift_coord_window(W)
     WINDOW *W;
{
  double TEMP1, TEMP2;

  TEMP1 = W->YMIN;
  TEMP2 = W->YMAX;
  W->YMIN = W->ZMIN;
  W->YMAX = W->ZMAX;
  W->ZMIN = TEMP1;
  W->ZMAX = TEMP2;
} /* end shift_coord_window */



void translate_line(L,X,Y,Z)

          LINE *L;
          double X,Y,Z;
{
          L->X1 += X;
          L->X2 += X;
          L->Y1 += Y;
          L->Y2 += Y;
          L->Z1 += Z;
          L->Z2 += Z;

}   /* END TRANSLATE_LINE  */



void translate_window(W,X,Y,Z)

          WINDOW *W;
          double X,Y,Z;
{
          W->XMIN += X;
          W->XMAX += X;
          W->YMIN += Y;
          W->YMAX += Y;
          W->ZMIN += Z;
          W->ZMAX += Z;

}   /* END TRANSLATE_WINDOW  */
```

```c
/* rotate about the vertical axis */

void rot_z(L,ORIENT)

        LINE *L;
        double ORIENT;
(
    double X1=L->X1,
            X2=L->X2,
            Y1=L->Y1,
            Y2=L->Y2,
        RADS = ORIENT * PI / 180.0 ;    /* convert degs to rads */
        L->X1 = X1*cos(RADS)-Y1*sin(RADS);
        L->X2 = X2*cos(RADS)-Y2*sin(RADS);
        L->Y1 = Y1*cos(RADS)+X1*sin(RADS);
        L->Y2 = Y2*cos(RADS)+X2*sin(RADS);


)   /* end rot_z */

void rot_window(W,ORIENT)

        WINDOW *W;
      double ORIENT;
(
      double XMIN=W->XMIN,
            XMAX=W->XMAX,
            YMIN=W->YMIN,
            YMAX=W->YMAX,
        RADS = ORIENT * PI / 180.0 ;    /* convert degs to rads */

        W->XMIN = XMIN*cos(RADS)-YMIN*sin(RADS);
        W->XMAX = XMAX*cos(RADS)-YMAX*sin(RADS);
        W->YMIN = YMIN*cos(RADS)+XMIN*sin(RADS);
        W->YMAX = YMAX*cos(RADS)+XMAX*sin(RADS);


)   /* end rot_z */




void perspective_transform(L,ZMIN)
        LINE *L;
      double ZMIN;
(
      double W1=L->Z1/ZMIN ,W2=L->Z2/ZMIN;

  if (W1!=0.0) (
        L->X1=L->X1/W1;
        L->Y1=L->Y1/W1;
```

```c
                    L->Z1=L->Z1/W1;
        }
    else
            printf("\nERROR --- tried to divide by W1=0\n");
    if (W2!=0.0) {
                L->X2=L->X2/W2;
                L->Y2=L->Y2/W2;
                L->Z2=L->Z2/W2;
        }
    else
            printf("\nERROR --- tried to divide by W2=0\n");
}    /* end perspective_transform */




void get_clipping_codes(L,ZMIN)
    LINE *L;
    double ZMIN;
{
    int i;

    for (i=0;i<=5;++i) {
      L->CLIP1[i]=0;
      L->CLIP2[i]=0;
    }
    if (L->Y1>-L->Z1)
      L->CLIP1[0]=1;
    if (L->Y1<L->Z1)
      L->CLIP1[1]=1;
    if (L->X1>-L->Z1)
      L->CLIP1[2]=1;
    if (L->X1<L->Z1)
      L->CLIP1[3]=1;
    if (L->Z1<-1.0)
      L->CLIP1[4]=1;
    if (L->Z1>ZMIN)
      L->CLIP1[5]=1;
    if (L->Y2>-L->Z2)
      L->CLIP2[0]=1;
    if (L->Y2<L->Z2)
      L->CLIP2[1]=1;
    if (L->X2>-L->Z2)
      L->CLIP2[2]=1;
    if (L->X2<L->Z2)
      L->CLIP2[3]=1;
    if (L->Z2<-1.0)
      L->CLIP2[4]=1;
    if (L->Z2>ZMIN)
      L->CLIP2[5]=1;
/*print_line(L);
```

```c
printf("\nclipping code1 = %d %d %d %d %d %d\n",L->CLIP1[0],L->CLIP1[1],
       L->CLIP1[2],L->CLIP1[3],L->CLIP1[4],L->CLIP1[5]);
printf("\nclipping code2 = %d %d %d %d %d %d\n",L->CLIP2[0],L->CLIP2[1],
       L->CLIP2[2],L->CLIP2[3],L->CLIP2[4],L->CLIP2[5]);*/
  }  /* end get_clipping_codes */


void clipt(NUM,DENOM,TE,TL)
     double NUM, DENOM;
     double *TE, *TL;
{
  double t;

/*printf("\nNUM= %.6lf  DENOM= %.6lf TE= %.6lf  TL=
%.6lf",NUM,DENOM,*TE,*TL);*/
  if (DENOM<0.0) {
    t=NUM/DENOM;
/*printf("    t= %.6lf",t);*/
    if (t>*TL)
      t=t;
/*      printf("\nclipt error1");*/
    else
      if (t>*TE)
          *TE=t;
  }
  if (DENOM>0.0) {
    t=NUM/DENOM;
/*printf("t= %.6lf",t);*/
    if (t<*TE)
      t=t;
/*      printf("\nclipt error2");*/
    else
      if (t<*TL)
          *TL=t;
  }
/*  if (DENOM==0.0)
    printf("\nclipt error #3 --> dividing by 0.0");*/
}   /* end clipt */


void clip_line(L,ZMIN)
     LINE *L;
     double ZMIN;
{
  double dx, dy, dz;
  double TMIN=0.0, TMAX=1.0;


  dx=L->X2-L->X1;
  dy=L->Y2-L->Y1;
  dz=L->Z2-L->Z1;
```

```c
/*printf("\ntmin= %.6lf    tmax= %.6lf    zmin = %.6lf",TMIN,TMAX,ZMIN);*/

  clipt((-L->X1-L->Z1),(dx+dz),&TMIN,&TMAX);
  clipt'.L->X1-L->Z1),(-dx+dz),&TMIN,&TMAX);
  clint (L->Y1-L->Z1),(-dy+dz),&TMIN,&TMAX);
  clipt((-L->Y1-L->Z1),(dy+dz),&TMIN,&TMAX);
  clipt((-L->Z1+ZMIN),(dz),&TMIN,&TMAX);
  clipt((-L->Z1-1),(-dz),&TMIN,&TMAX);
  if (TMAX<1) {
    L->X2 = L->X1 + (TMAX*dx);
    L->Y2 = L->Y1 + (TMAX*dy);
    L->Z2 = L->Z1 + (TMAX*dz);
  }
  if (TMIN>0) {
    L->X1 = L->X1 + (TMIN*dx);
    L->Y1 = L->Y1 + (TMIN*dy);
    L->Z1 = L->Z1 + (TMIN*dz);
  }
}   /* end clip_line */




/* returned codes: 0 outside of view volume
                   1 partially inside volume
                             2 entirely in view volume
*/

int clip_line_3d(L)
         LINE *L;
{
         int          IN_VOLUME=1, i, C1=0 ,C2=0;

         for (i=0;i<=5;++i) {
           C1+=L->CLIP1[i];
           C2+=L->CLIP2[i];
           if ((L->CLIP1[i]==1)&&(L->CLIP2[i]==1))
             IN_VOLUME=0;   /* outside view volume */
         }
         if ((IN_VOLUME==1)&&((C1==0)&&(C2==0)))
           IN_VOLUME=2;   /* entirely in view volume */
         return IN_VOLUME;
}   /* end clip_line_3d */




void display_window(W)
     WINDOW *W;
{
  int DUMMY;
```

```c
printf("\n\nWindow limits calculated: ");
printf("\nX: %.21f-%.21f\nY: %.21f-%.21f\nZ: %.21f-%.21f\n\n",
            W->XMIN,W->XMAX,W->YMIN,W->YMAX,W->ZMIN,W->ZMAX);
fflush(stdout);
printf("\n\nEnter a number to continue");
/*scanf("%d",&DUMMY);*/
}

double myabs(X)
      double X;
{
  if (X<0.0)
    X=0.0-X;
  return X;
}

void map_to_screen(L,XMIN,YMIN)
      LINE *L;
      double XMIN,YMIN;
{
  L->X1 = myabs((L->X1-XMIN)/(2*XMIN)*MAX_X);
  L->X2 = myabs((L->X2-XMIN)/(2*XMIN)*MAX_X);
  L->Y1 = myabs((L->Y1-YMIN)/(2*YMIN)*MAX_Y);
  L->Y2 = myabs((L->Y2-YMIN)/(2*YMIN)*MAX_Y);

/* 1279.0 , 1023.0 */
}  /* end map_to_screen */


/* retrun with 1 if line was not totally clipped out of view */

int project_line(X,Y,Z,ORIENT,L,W,W1,FL)

            double X,Y,Z,ORIENT;
            LINE *L;
            WINDOW *W,*W1;
        double FL;
{
            double ZMIN,SCALEX,SCALEY,SCALEZ,VRP_Z;
            int    USED_LINE=1, CLIPT;
            double fl=1.24;
            double X1,Y1,Z1,XTEMP,YTEMP,RADS;


            translate_line(L,-W->XMIN,-W->YMIN,-W->ZMIN);
            rot_z(L,-ORIENT);

X1=X-W->XMIN;
Y1=Y-W->YMIN;
```

102

```c
Z1=Z-W->ZMIN;
XTEMP=X1;
YTEMP=Y1;
RADS = ORIENT * PI / 180.0 ;    /* convert degs to rads */
X1 = XTEMP*cos(-RADS)-YTEMP*sin(-RADS);
Y1 = YTEMP*cos(-RADS)+XTEMP*sin(-RADS);


        translate_line(L,-X1,-Y1,-Z1);
/* change from world to view coords */


/* shear so view volume centered on z-axis */


/* now scale view vol to unity using s_per */
/* NOTE: FAR_CLIP is global value */
        VRP_Z = -Y1; /*since still in world coords*/
        SCALEX = 2.0*VRP_Z/((W1->XMAX-W1->XMIN)*(VRP_Z+FARCLIP));
        SCALEY = 2.0*VRP_Z/((W1->YMAX-W1->YMIN)*(VRP_Z+FARCLIP));
        SCALEZ = -1.0/(VRP_Z+FARCLIP);
        shift_coord_line(L);
        scale_line(L,SCALEX,SCALEY,SCALEZ);
/*printf("\nafter scaling: ");
print_line(L);*/
        ZMIN=SCALEZ*(VRP_Z+NEARCLIP);
/*printf("\nZMIN = %.5lf",ZMIN);*/
        get_clipping_codes(L,ZMIN);
/* divide by w/d */
        CLIPT=clip_line_3d(L);
        if (CLIPT!=0) {
                if (CLIPT==1)
                        clip_line (L,ZMIN);
/*printf("\nafter clipping:");
print_line(L);*/
                        perspective_transform(L,ZMIN);
/*printf("\nafter M-per");
                        print_line(L);*/
                        map_to_screen(L,ZMIN,ZMIN);
/*printf("\nafter map to screen\n\n");
                        print_line(L);*/


        }
        else
                USED_LINE=0;
        return USED_LINE;
}    /* end project_line */

void remove_line(L,LH)
        LINE *L;
        LINE_HEAD *LH;
{
        LINE *NEXT_L=LH->LINE_LIST, *TRASH;
```

```
                if (L==LH->LINE_LIST) {
                        LH->LINE_LIST=LH->LINE_LIST->NEXT;
                        free(L);
                }
                else {
                        while ((NEXT_L->NEXT)&&(NEXT_L->NEXT!=L)) {
                                NEXT_L=NEXT_L->NEXT;
                        }
                        NEXT_L->NEXT=NEXT_L->NEXT->NEXT;
                        free(L);
                }
                LH->LINES--;
}  /* end remove_line */




void remove_vert_line(L,LH)
                LINE *L;
                LINE_HEAD *LH;
{
                LINE *NEXT_L=LH->VLINE_LIST, *TRASH;

                if (L==LH->VLINE_LIST) {
                        LH->VLINE_LIST=LH->VLINE_LIST->NEXT;
                        free(L);
                }
                else {
                        while ((NEXT_L->NEXT)&&(NEXT_L->NEXT!=L)) {
                                NEXT_L=NEXT_L->NEXT;
                        }
                        NEXT_L->NEXT=NEXT_L->NEXT->NEXT;
                        free(L);
                }
                LH->VERT_LINES--;
}  /* end remove_vert_line */




LINE_HEAD *get_view(PRPX,PRPY,PRPZ,ORIENT,W,FL)

                double PRPX,PRPY,PRPZ,ORIENT,FL;
                WORLD *W;
{
                LINE            *NEXT_L,*TRASH;
                LINE_HEAD   *LH;
                WINDOW      *WIN=calc_window(PRPX,PRPY,PRPZ,ORIENT,FL);
                WINDOW      *W1=calc_window(PRPX,PRPY,PRPZ,ORIENT,FL);
                double      Z1, Z2, temp, XX, YY, ZZ, RADS, XTEMP, YTEMP;
                int         count=0;

                translate_window(W1,-(WIN->XMIN),-(WIN->YMIN),
```

```
                                    -(WIN->ZMIN));
                rot_window(W1,-ORIENT);

XX=PRPX-WIN->XMIN;
YY=PRPY-WIN->YMIN;
ZZ=PRPZ-WIN->ZMIN;
XTEMP=XX;
YTEMP=YY;
RADS = ORIENT * PI / 180.0 ;
XX = XTEMP*cos(-RADS)-YTEMP*sin(-RADS);
YY = YTEMP*cos(-RADS)+XTEMP*sin(-RADS);


                translate_window(W1,-XX,-YY,-ZZ);
/* change from world to view coords */
                shift_coord_window(W1);
                LH=conduct_visibility_sweep(W,PRPX,PRPY,PRPZ);
                NEXT_L=LH->VLINE_LIST;
                while (NEXT_L) {
/*printf("\nRAW LINE:");
lprint_l(NEXT_L);*/
                        if
(project_line(PRPX,PRPY,PRPZ,ORIENT,NEXT_L,WIN,W1,FL)!=1) {
                                    TRASH=NEXT_L;
                                    NEXT_L=NEXT_L->NEXT;
                                    remove_vert_line(TRASH,LH);
                        }
                        else         {
/*lprint_l(NEXT_L);*/
                                    NEXT_L=NEXT_L->NEXT;
                        }
                } /* end while */
                NEXT_L=LH->LINE_LIST;
                while (NEXT_L) {
                        if
(project_line(PRPX,PRPY,PRPZ,ORIENT,NEXT_L,WIN,W1,FL)!=1) {
                                    TRASH=NEXT_L;
                                    NEXT_L=NEXT_L->NEXT;
                                    remove_line(TRASH,LH);
                        }
                        else         {
                                    NEXT_L=NEXT_L->NEXT;
                        }
                } /* end while */

/*
printf("\nCompleted projection\n");
printf("\n\nVert lines kept = %d\nOthers = %d\n",LH->VERT_LINES,LH-
>LINES);
fflush(stdout);
*/
                free(WIN);
```

```
            free(W1);
            return LH;
)    /* end get_view */




LINE_HEAD *get_view_orig(PRPX,PRPY,PRPZ,ORIENT,W,FL)

            double PRPX,PRPY,PRPZ,ORIENT,FL;
            WORLD *W;
(

            POLYHEDRON *NEXT_PH;


            POLYGON    *NEXT_PG;
            VERTEX     *NEXT_V;
        INSTANCE    *NEXT_I;
            LINE           *NEXT_L;
            LINE_HEAD  *LH=create_line_head();
            WINDOW     *WIN=calc_window(PRPX,PRPY,PRPZ,ORIENT,FL);
            WINDOW     *W1=calc_window(PRPX,PRPY,PRPZ,ORIENT,FL);
            double     Z1, Z2, temp, XX, YY, ZZ, RADS, XTEMP, YTEMP;
            int           count=0;

            translate_window(W1,-(WIN->XMIN),-(WIN->YMIN),
                                      -(WIN->ZMIN));
            rot_window(W1,-ORIENT);

XX=PRPX-WIN->XMIN;
YY=PRPY-WIN->YMIN;
ZZ=PRPZ-WIN->ZMIN;
XTEMP=XX;
YTEMP=YY;
RADS = ORIENT * PI / 180.0 ;
XX = XTEMP*cos(-RADS)-YTEMP*sin(-RADS);
YY = YTEMP*cos(-RADS)+XTEMP*sin(-RADS);

            translate_window(W1,-XX,-YY,-ZZ);
/* change from world to view coords */
            shift_coord_window(W1);

            NEXT_PH=W->POLYHEDRON_LIST;
            while (NEXT_PH) (
                NEXT_I=NEXT_PH->INSTANCE_LIST;
                while (NEXT_I) (
                        NEXT_PG=NEXT_PH->POLYGON_LIST;
                        while(NEXT_PG) (
                                NEXT_V=NEXT_PG->VERTEX_LIST;
                                Z1=NEXT_PG->Z_VALUE;
                                while(NEXT_V) (
                                    if (NEXT_V->VERT_EDGE) (
```

106

```c
                                              Z2=find_z(NEXT_PH,NEXT_V-
>VERT_EDGE);
                                NEXT_L=make_line(NEXT_I,NEXT_V,NEXT_V-
>VERT_EDGE,Z1,Z2);

printf("\nRAW LINE:");
lprint_l(NEXT_L);
count++;
                   if
(project_line(PRPX,PRPY,PRPZ,ORIENT,NEXT_L,WIN,W1,FL)==1) {

printf("\nACCEPTED\n");lprint_l(NEXT_L);


add_lines(LH,NEXT_L);
                         }
                                      }  /* end if */
                                      if (NEXT_V->NEXT) {

NEXT_L=make_line(NEXT_I,NEXT_V,NEXT_V->NEXT,Z1,Z1);
printf("\nRAW LINE:");
lprint_l(NEXT_L);
count++;
                   if
(project_line(PRPX,PRPY,PRPZ,ORIENT,NEXT_L,WIN,W1,FL)==1) {

printf("\nACCEPTED\n");
lprint_l(NEXT_L);

add_lines(LH,NEXT_L);
                         }
                                         NEXT_V=NEXT_V->NEXT;
                                      }  /* end if */
                                      else {
                          NEXT_L=make_line(NEXT_I,NEXT_V,NEXT_PG-
>VERTEX_LIST,Z1,Z1);
printf("\nRAW LINE:");
lprint_l(NEXT_L);
count++;
                   if
(project_line(PRPX,PRPY,PRPZ,ORIENT,NEXT_L,WIN,W1,FL)==1) {

printf("\nACCEPTED\n");
lprint_l(NEXT_L);

add_lines(LH,NEXT_L);
                         }
                                         NEXT_V=NULL;
                                      }  /* end else */
                                   }   /* end while */
                                   NEXT_PG=NEXT_PG->NEXT;
```

107

```
                              )    /* end while */
                         NEXT_I=NEXT_I->NEXT;
                  )    /* end while */
                NEXT_PH=NEXT_PH->NEXT;
              )    /* end while */
/*            print_line_list(LH);
printf("\n\nTotal lines considered = %d\n\n",count);
printf("\n\nTotal lines accepted = %d\n\n",LH->LINES);*/
printf("\n\nLines returned by get_view_orig\n\n");
lprint_llist(LH);
              return LH;
)    /* end get_view_orig */
```

```
/*
FILE NAME:    visibility.h
AUTHOR:       James Stein
PROJECT:      Thesis, supporting Yamabico vision system
DATE:         March 1992
ADVISOR:      Dr. Kanayama

COMMENTS: This file implements a algorithm which determines the set of
visible line seen from a given position in a wire frame model. The observer
is
assumed to have omni-directional vision. To impose the physical limits of
a camera's field of view, the function get_view in file graphics.h can be
sent a model (as it in turn uses this file).

Primary Function(s):

        - conduct_vsiibility_sweep

                INPUT:    W                  a pointer to a 2d+ world model
                          EYE_X,EYE_Y,EYE_Z  position of observer in model W

                OUTPUT:   LINE_LIST          structure pointing to 2 lists of
                                             visible vertical and non-vertical
lines
*/




/*---------------------- Structure definitions:-----------------------
----*/


typedef struct sweep_link {
        double THETA, X, Y, Z,
               MIN_Z, MAX_Z, UPPER_Z, DIST;
        VERTEX *V;
        INSTANCE *I;
        POLY_LINK *CEILINGS;
        struct sweep_link *PREV, *NEXT;
} SWEEP_LINK;


/*------------------------------------------------------------------*/


typedef struct considered_link {
        double MIN_SWEEP,
                       MIN_Z, MAX_Z, DIST,
                       NEW_MIN_Z,NEW_MAX_Z,UPPER_Z;
                int
VISIBILITY,B_VISIBILITY,NEW_VISIBILITY,NEW_B_VISIBILITY;
```

109

```c
            POLY_LINK *CEILINGS;
            SWEEP_LINK *SL1, *SL2;
            struct considered_link *NEXT;
) CONSIDERED_LINK;


/*----------------------------------------------------------------*/


typedef struct considered_head {
            CONSIDERED_LINK *LINKS;
) CONSIDERED_HEAD;

/*----------------------------------------------------------------*/


/* global variables: */

static double X,Y,Z;   /*Position of observer within the model*/
static double THETA;   /*Current angle of visibility sweep*/
int IN_MAIN;           /*if 0 we are still preprocessing straddlers*/



void line_ray_intersection(CONSIDERED_LINK *CL,double ANGLE,
                                    double *INT_X,double *INT_Y,double
*DIST);


/* Doubles can be truncated to 4 decimal places to compensate for
inexactness
    of floating point operations*/

double trunc(X)
     double X;
{
  int DUMMY;
  double XX=X;

  DUMMY=XX*10000;
  XX=DUMMY;
  XX=XX/10000.0;
  return XX;
}

/*****************************CONVERSION
FUNCTIONS********************/

double degs(RADS)
     double RADS;
{
```

110

```c
  return trunc(RADS*180.0/PI);
}



double rads(DEGS)
     double DEGS;
{
  return trunc(DEGS*PI/180.0);
}


/
*********************************************************************
/


/* Determines if the edges from 2 considered links are colinear*/

int colinear(F,B)
           CONSIDERED_LINK *F,*B;
{
           double M1,M2;   /*we will compare slopes and distance*/

           M1=trunc((F->SL1->Y-F->SL2->Y)/(F->SL1->X-F->SL2->X));
           M2=trunc((B->SL1->Y-B->SL2->Y)/(B->SL1->X-B->SL2->X));
           if ((M1==M2)&&(F->DIST==B->DIST))
                      return 1;
           else
                      return 0;
}



/*****************************COUNTER CLOCKWISE
CHECKS*********************/

int ccw(SL,PREV_SL)
           SWEEP_LINK *SL, *PREV_SL;
{
           double AREA;

           AREA= 0.5*((SL->X-X)*(PREV_SL->Y-Y)-
                      (PREV_SL->X-X)*(SL->Y-Y));
           if (AREA>0.0)
              return 1;
           else
              return 0;
}    /* end ccw */



int ccw2(SL1,SL2,SL3)
```

```
                SWEEP_LINK *SL1,*SL2,*SL3;
{
                double AREA;

                AREA= 0.5*((SL2->X-SL1->X)*(SL3->Y-SL1->Y)-
                        (SL3->X-SL1->X)*(SL2->Y-SL1->Y));
                if (AREA>0.0)
                   return 1;
                else
                        return 0;
}   /* end ccw */


/
*********************************************************************
*****/


/* Finds the angle from X1,Y1 to V for use in determining if X1,Y1 lies
   within the bounds of a polygon.*/

double find_theta(X1,Y1,V,I)
                double X1,Y1;
                VERTEX *V;
                INSTANCE *I;
{
                double X2,Y2,T;
                double LOCAL_X,LOCAL_Y,ROT_X,ROT_Y,RADS;

                LOCAL_X = V->X - I->PIVOT_X;
                LOCAL_Y = V->Y - I->PIVOT_Y;

 /* rotate about the z axis */
                RADS = I->ROTATION * PI / 180.0 ;   /* convert degs to rads */
                ROT_X = (cos(RADS)*LOCAL_X)+(sin(RADS)*LOCAL_Y);
                ROT_Y = (cos(RADS)*LOCAL_Y)-(sin(RADS)*LOCAL_X);

  /* translate to proper position in world model */

                X2 = I->X + ROT_X;
                Y2 = I->Y + ROT_Y;
                if ((X1==X2)||((Y1==Y2)&&(X1==X2)))
                        T=0.0;
                else
                        T=atan2(Y2-Y1,X2-X1);   /* both won't be 0 */
                if (T<0.0)
                        T+=rads(360.0); /* normalize to 0-360 */
                return T;
}  /* end find_theta */
```

```
/* This function determines if the point X1,Y1 lies within the polygon, PG.
   The angle formed between lines drawn to each edge of PG is calculated.
   CW angles are added and CCW ones are subtracted from the sum.
   If the sum is not equal to 0.0 the point is within PG and 1 is
returned.*/


int in_polygon(X1,Y1,PG,I)
            double X1,Y1;
            POLYGON *PG;
            INSTANCE *I;
{
            VERTEX *FIRST_V, *V=PG->VERTEX_LIST;
            double THETA1,THETA2,FIRST_THETA,SUM=0.0,SUM1=0.0;
            double XX,YY;


            THETA2=find_theta(X1,Y1,V,I);
            FIRST_V=V;
            FIRST_THETA=THETA2;
            while (V->NEXT) {
                        if ((X1==V->X)&&(Y1==V->Y))
                                    SUM1=1.0;           /*if directly under
a point accept*/
                        THETA1=THETA2;
                        THETA2=find_theta(X1,Y1,V->NEXT,I);
/*ccw*/
                        if ((0.5*((V->X-X1)*(V->NEXT->Y-Y1)-
                            (V->NEXT->X-X1)*(V->Y-Y1)))>0.0) {
                                    if (THETA2<THETA1)

SUM+=(THETA2+rads(360.0))-THETA1;
                                    else
                                                SUM+=THETA2-THETA1;
                        }
/*cw*/                  else {
                                    if (THETA2>THETA1)
                                                SUM+=THETA2-
(THETA1+rads(360.0));
                                    else
                                                SUM+=THETA2-THETA1;
                        }
                        V=V->NEXT;
            } /* end while */
/*Lastly: check the closing edge to see if we add or subtract its angle*/
            THETA1=THETA2;
            THETA2=FIRST_THETA;
            if ((0.5*((V->X-X1)*(FIRST_V->Y-Y1)-(FIRST_V->X-X1)*(V->Y-
Y1)))>
                  0.0) {   /*ccw*/
                        if (THETA2<THETA1)
                                    SUM+=(THETA2+rads(360.0))-THETA1;
```

```c
                        else
                                SUM+=THETA2-THETA1;
        }
        else {
                if (THETA2>THETA1)
                        SUM+=THETA2-(THETA1+rads(360.0));
                else
                        SUM+=THETA2-THETA1;
        }
        if (((trunc(SUM)==0.0))&&(SUM1==0.0))
                return 0;
        else
                return 1;
}


/* Function checks to see which ceiling of CL's ceiling list the 1st
endpoint
   falls under. This height is returned and is used to determine how much
   coverage the CL has along the z-axis (that is what angle bound the
portion
   of the z-axis which CL occludes*/

double find_ceiling_z(CL)
        CONSIDERED_LINK *CL;
{
        double IX,IY,DIST,CEILING_Z_VALUE=(-9999999.9);
        POLY_LINK *NEXT_C=CL->CEILINGS;
        int FOUND=0;

        IX=CL->SL1->X;
        IY=CL->SL1->Y;
        while (NEXT_C) {
/*keep track of highest ceiling over CL*/
        if ((in_polygon(IX,IY,NEXT_C->REF_POLY,CL->SL1->I)==1)&&
            (NEXT_C->REF_POLY->Z_VALUE+CL->SL1->I-
>Z>CEILING_Z_VALUE)) {
                        CEILING_Z_VALUE=NEXT_C->REF_POLY->Z_VALUE+CL-
>SL1->I->Z;
                        FOUND=1;
        }
           NEXT_C=NEXT_C->NEXT;
        }
        if (FOUND==0) {
           CEILING_Z_VALUE=CL->SL1->UPPER_Z;   /*if none ht same as
endpoint*/
        }
        return trunc(CEILING_Z_VALUE);      /*return highest ceiling
ht*/
}  /* end find_ceiling_z */
```

```
/* Calculate the minimum and maximum angles which SL covers on the z-axis.
   Any object which is farther away and behind SL that falls within these
   limits will not be able to be seen. */

void calc_z_coverage(SL)
            SWEEP_LINK *SL;
{
            double dz,LEN;

            dz=SL->Z-Z;
            LEN=SL->DIST;   /*dist to line in X-Y plane*/
            if (LEN==0)
                        LEN=0.00001;
            SL->MIN_Z=trunc(atan(dz/LEN));
            dz=SL->UPPER_Z-Z;
            SL->MAX_Z=trunc(atan(dz/LEN));
}   /* end calc_z_coverage */


/*  Absolute value of a double */

double my_abs(A)
            double A;
{
            if (A>=0.0)
                        return A;
            else
                        return -A;
}


/* Calculates the limiting angles along the z-axis for each item on the
   considered list. These limits are based upon the height of each
   endpoint (value of CL->MIN_Z) and the height of the ceiling (if any)
   lying above CL (CL->UPPER_Z). */

void calc_current_z_coverage(CLIST)
            CONSIDERED_HEAD *CLIST;
{
            CONSIDERED_LINK *CL=CLIST->LINKS;
            double MIN,MAX,DIST;
            double dx,dy,dz,IX,IY,LEN1,LEN2,
                        CEILING_Z;

    while (CL) {
            CL->NEW_MAX_Z=trunc(atan((CL->UPPER_Z-Z)/CL->DIST));
            CL->NEW_MIN_Z=trunc(atan((CL->SL1->Z-Z)/CL->DIST));
            CL->NEW_VISIBILITY=1;    /*reset visibilities*/
            CL->NEW_B_VISIBILITY=1;
```

115

```
                        CL=CL->NEXT;
        }
}   /* end calc_current_z_coverage */



/**********************MEMORY ALLOCATION
FUNCTIONS*************************/

LINE_HEAD *make_line_head()
{
                LINE_HEAD *LH=(LINE_HEAD *)malloc(sizeof(LINE_HEAD));

                LH->LINES=0;
                LH->VERT_LINES=0;
                LH->LINE_LIST=NULL;
                LH->TAIL=NULL;
                LH->VLINE_LIST=NULL;
                LH->VTAIL=NULL;
                return LH;
}    /* end make_line_head */




CONSIDERED_HEAD *make_considered_head()
{
                CONSIDERED_HEAD *CH;

                CH= (CONSIDERED_HEAD *)malloc(sizeof(CONSIDERED_HEAD));
                CH->LINKS=NULL;
                return CH;
}  /* end make_considered_head */




SWEEP_LINK *make_sweep_link(PH,PG,V,I,PG_Z)
                POLYHEDRON *PH;
                POLYGON *PG;
                VERTEX *V;
                INSTANCE *I;
                double PG_Z;
{
                SWEEP_LINK *SL;
                double LOCAL_X,LOCAL_Y,ROT_X,ROT_Y,RADS;

                SL= (SWEEP_LINK *)malloc(sizeof(SWEEP_LINK));
                SL->PREV= NULL;
                SL->NEXT= NULL;
                SL->V=V;
```

116

```c
                SL->I=I;
                SL->CEILINGS=PG->CEILING_LIST;
                LOCAL_X = V->X - I->PIVOT_X;
                LOCAL_Y = V->Y - I->PIVOT_Y;

   /* rotate about the z axis */
                RADS = I->ROTATION * PI / 180.0 ;   /* convert degs to rads */
                ROT_X = (cos(RADS)*LOCAL_X)+(sin(RADS)*LOCAL_Y);
                ROT_Y = (cos(RADS)*LOCAL_Y)-(sin(RADS)*LOCAL_X);

    /* translate to proper position in world model */

                SL->X = trunc(I->X + ROT_X);   /*must be truncated*/
                SL->Y = trunc(I->Y + ROT_Y);
                SL->Z = trunc(I->Z + PG_Z);

                SL->THETA=(atan2(SL->Y-Y,SL->X-X));   /* both won't be 0 */
                if (SL->THETA<0.0)
                        SL->THETA=(2.0*PI+SL->THETA); /* normalize to 0-
360 */
                SL->DIST= trunc(sqrt(pow((SL->Y-Y),2.0)+pow((SL->X-
X),2.0)));
                if (V->VERT_EDGE)
                        SL->UPPER_Z=find_z(PH,V->VERT_EDGE)+I->Z;
                else
                        SL->UPPER_Z=SL->Z;
                if ((PH->OBSTACLE==0)&&(PG->FLOOR==0))
                        SL->UPPER_Z=99999999999.9; /*max float to cover
90 degs*/
                calc_z_coverage(SL);
                return SL;
}   /* end make_sweep_link */




CONSIDERED_LINK *make_considered_link(SL)
                SWEEP_LINK *SL;
{
                CONSIDERED_LINK *CL=(CONSIDERED_LINK
*)malloc(sizeof(CONSIDERED_LINK));

                CL->SL1=SL;
                CL->DIST=SL->DIST;
                CL->SL2=SL->PREV;
                CL->CEILINGS=SL->CEILINGS;
                CL->VISIBILITY=1;
                CL->B_VISIBILITY=1;
                CL->NEW_VISIBILITY=1;
                CL->NEW_B_VISIBILITY=1;
                CL->NEXT=NULL;
```

```
                CL->MIN_SWEEP=SL->THETA;/*set  min to reflect sweep so far*/
                CL->MIN_Z=SL->MIN_Z;
                CL->MAX_Z=SL->MAX_Z;
                if (CL->SL1->UPPER_Z>9999999.9)    /*need to trunc????*/
                            CL->UPPER_Z=99999999999.9;
                else
                            CL->UPPER_Z=find_ceiling_z(CL);
                return CL;
) /* make_considered_link */


/
*************************************************************************
**/




/***************************MEMORY
DEALLOCATION***************************/

void        free_sweep_list(SLIST)
            SWEEP_LINK *SLIST;
(

            SWEEP_LINK *TRASH=SLIST;

            while (TRASH) (
                        SLIST=SLIST->NEXT;
                        free(TRASH);
                        TRASH=SLIST;
            )
) /* end free_sweep_list */




void free_clist(CLIST)
            CONSIDERED_HEAD *CLIST;
(

            CONSIDERED_LINK *NEXT_CL=CLIST->LINKS,*TRASH;

            while (NEXT_CL) (
                        TRASH=NEXT_CL;
                        NEXT_CL=NEXT_CL->NEXT;
                        free(TRASH);
            )
            free(CLIST);
)  /* end free_clist */


/
*************************************************************************
**/
```

```c
/**************************DISPLAY
FUNCTIONS******************************
  NOTE: These functions were used in debugging, but they have been left
       in case inspection of intermediate results is needed in the future.*/


void print_l(L)
          LINE *L;
(
          printf("\n\nline: X1=%.2lf Y1=%.2lf Z1=%.2lf ",L->X1,L->Y1,L-
>Z1);
          printf("\n      X2=%.2lf Y2=%.2lf Z2=%.2lf \n",L->X2,L->Y2,L-
>Z2);
          fflush(stdout);
}


void print_llist(LIST)
          LINE_HEAD *LIST;
(
          LINE *NEXT_L=LIST->VLINE_LIST;

          printf("\n\n\nVertical lines (%d) are:\n\n",LIST-
>VERT_LINES);
          while (NEXT_L) (
                    print_l(NEXT_L);
                    NEXT_L=NEXT_L->NEXT;
          )
          printf("\n\n\nnon-vertical lines (%d) are:\n\n",LIST-
>LINES);
          fflush(stdout);
          NEXT_L=LIST->LINE_LIST;
          while (NEXT_L) (
                    print_l(NEXT_L);
                    NEXT_L=NEXT_L->NEXT;
          )
}




void print_sl(SL)
          SWEEP_LINK *SL;
(
          printf("\nSL:  X=%.2lf Y=%.2lf Z=%.2lf",SL->X,SL->Y,SL->Z);
          printf("\n     THETA=%.20lf DIST=%.2lf",degs(SL->THETA),SL-
>DIST);
          printf("\n     MIN_Z=%.2lf MAX_Z=%.2lf",
                    degs(SL->MIN_Z),degs(SL->MAX_Z));
          if (SL->PREV==NULL)
                    printf("\nWarning no previous link");
```

119

```c
        if (SL->NEXT==NULL)
                printf("Warning should be last link");
        fflush(stdout);

} /* end print_sl*/




void print_slist(SL)
        SWEEP_LINK *SL;
{
        SWEEP_LINK *NEXT_SL=SL;

        printf("\n\nSWEEP LIST:\n\n");
        while (NEXT_SL) {
                print_sl(NEXT_SL);
                NEXT_SL=NEXT_SL->NEXT;
        }
}




void print_cl(CL)
        CONSIDERED_LINK *CL;
{

        printf("\n\n    MIN_Z=%.21f MAX_Z=%.21f",
                degs(CL->MIN_Z),degs(CL->MAX_Z));
        printf("\n\n NEW_MIN_Z=%.21f NEW_MAX_Z=%.21f",
                degs(CL->NEW_MIN_Z),degs(CL->NEW_MAX_Z));
        printf("\n    MIN_SWEEP=%.21f  DIST=%.21f",
                degs(CL->MIN_SWEEP),CL->DIST);
        printf("\nUPPER_Z:%.21f",CL->UPPER_Z);
        printf("\nOLD: VISIBLE=%d  B_VISIBLE=%d",CL->VISIBILITY,CL->B_VISIBILITY);
        printf("\nNEW: VISIBLE=%d  B_VISIBLE=%d",
                CL->NEW_VISIBILITY,CL->NEW_B_VISIBILITY);
        print_sl(CL->SL1);
        print_sl(CL->SL2);
}  /* end print_cl */




void print_clist(CLIST)
        CONSIDERED_HEAD *CLIST;
{
        CONSIDERED_LINK *CL=CLIST->LINKS;

        printf("\n\nConsidered list (THETA=%.21f):\n\n",degs(THETA));
```

```
                while (CL) {
                        print_cl(CL);
                        CL=CL->NEXT;
                }
) /* end print_clist */

/
*************************************************************************
**/


/* Sweep links are added to the list in order of their THETA values*/

SWEEP_LINK *add_sweep_link(LIST,LINK)
             SWEEP_LINK *LIST, *LINK;
{
             SWEEP_LINK *TEMP;

             if (LIST) {
                TEMP=LIST;
                if (TEMP->THETA>LINK->THETA) {
                        LINK->NEXT=LIST;
                        LIST=LINK;    /* inserted as 1st element */
                }
                else {
                        while ((TEMP->NEXT)&&(TEMP->NEXT->THETA<=LINK-
>THETA)) {
                                TEMP=TEMP->NEXT;
                        }
                        LINK->NEXT=TEMP->NEXT;
                        TEMP->NEXT=LINK;
                } /* end else */
             } /* end if */
             else
                LIST=LINK;    /* is first element to add to list */
             return LIST;
)   /* end add_sweep_link */



/*   This function scans through the entire world model (W). A sweep link
is
   made for each vertex of the model. The angle from the observer (global
   variable) to the vertex is calculated and used to sort the links.
   When a link is made, we also inspect its ->VERT_EDGE pointer to see
   if a vertical line leaves it. Calculate_z_coverage uses the height of
   this vertical line to determine coverage of the vertex along the z-axis.
     Each sweep link has its PREV pointer assigned to indicate the link
   which preceeded it in the polygon list. In latter processing only sweep
   links with a ccw relationship to this PREV link will be considered as
   visible.
```

121

Since we will latter require all floors residing above the observer and all ceiling below them to be visible, we inspect each polygon for these properties. When a polygon satifies one of these, it's vertices are processed a second time in reverse order. This ensures that every edge of the polygon will swho up as a ccw CONSIDERED_LINK.
*/

```
SWEEP_LINK *make_sweep_list(W)
          WORLD *W;
{
          SWEEP_LINK *SWEEP_LIST=NULL,*NEXT_L,*LAST_L,*FIRST_L;
          POLYHEDRON *NEXT_PH;
          POLYGON *NEXT_PG;
          VERTEX *NEXT_V, *LAST_V;
          INSTANCE *NEXT_I, *LAST_I;

    NEXT_PH=W->POLYHEDRON_LIST;
    while (NEXT_PH) {
       NEXT_I=NEXT_PH->INSTANCE_LIST;
       while (NEXT_I) {
             NEXT_PG=NEXT_PH->POLYGON_LIST;
             while (NEXT_PG) {
                 NEXT_V=NEXT_PG->VERTEX_LIST;
                 NEXT_L=make_sweep_link(NEXT_PH,NEXT_PG,NEXT_V,NEXT_I,
                                                  NEXT_PG->Z_VALUE);
                 SWEEP_LIST=add_sweep_link(SWEEP_LIST,NEXT_L); /*make and
add links*/
                 FIRST_L=NEXT_L;
                 LAST_L=NEXT_L;
                 NEXT_V=NEXT_V->NEXT;
                 while (NEXT_V) {

NEXT_L=make_sweep_link(NEXT_PH,NEXT_PG,NEXT_V,NEXT_I,
                                                        NEXT_PG-
>Z_VALUE);
                           NEXT_L->PREV=LAST_L;
                           SWEEP_LIST=add_sweep_link(SWEEP_LIST,NEXT_L);
                           LAST_L=NEXT_L;
                           NEXT_V=NEXT_V->NEXT;
                 } /* end while */
             FIRST_L->PREV=LAST_L;   /* add line which closes polygon */
/* Make entire polygon ccw so it may be visible */
                 if (((((NEXT_PG->Z_VALUE+NEXT_I->Z<Z)&&(NEXT_PG-
>FLOOR==0))||
                            ((NEXT_PG->Z_VALUE+NEXT_I->Z>Z)&&(NEXT_PG-
>FLOOR==1)))||
                            ((NEXT_PH->OBSTACLE==0)&&(NEXT_PG-
>FLOOR==0))) {
```

/* To cut down on processing time the above if statement can be commented out and the below one used. This has the effect of assuming a model

is composed of only large objects (observer doesn't look down or up to
them).
We still must make enclosure ceiling visible since items such as door jam
ceilings will not always be above the observe*/

```
/*              if ((NEXT_PH->OBSTACLE==0)&&(NEXT_PG->FLOOR==0)) (*/
                NEXT_V=NEXT_PG->VERTEX_LIST;
                NEXT_L=make_sweep_link(NEXT_PH,NEXT_PG,NEXT_V,NEXT_I,
                                               NEXT_PG->Z_VALUE);
                if (!((NEXT_PH->OBSTACLE==0)&&(NEXT_PG->FLOOR==0))) (
                  NEXT_L->MAX_Z=NEXT_L->MIN_Z; /*take away height if any*/
                  NEXT_L->CEILINGS==NULL;
                  NEXT_L->UPPER_Z=NEXT_L->Z;
                  )
                FIRST_L=NEXT_L;
                NEXT_V=NEXT_V->NEXT;
                while (NEXT_V) (

LAST_L=make_sweep_link(NEXT_PH,NEXT_PG,NEXT_V,NEXT_I,
                                                          NEXT_PG-
>Z_VALUE);
                if (!((NEXT_PH->OBSTACLE==0)&&(NEXT_PG->FLOOR==0))) (
                        LAST_L->MAX_Z=LAST_L->MIN_Z; /*take away height
if any*/
                        LAST_L->CEILINGS==NULL;
                        LAST_L->UPPER_Z=LAST_L->Z;
                )

                        NEXT_L->PREV=LAST_L;
                        SWEEP_LIST=add_sweep_link(SWEEP_LIST,NEXT_L);
                        NEXT_L=LAST_L;
                        NEXT_V=NEXT_V->NEXT;
                )
                NEXT_L->PREV=FIRST_L;
                SWEEP_LIST=add_sweep_link(SWEEP_LIST,NEXT_L);
                )
                NEXT_PG=NEXT_PG->NEXT;
            )   /* end while  NEXT_PG*/
          NEXT_I=NEXT_I->NEXT;
      ) /* end while NEXT_I */
      NEXT_PH=NEXT_PH->NEXT;
    ) /* end while NEXT_PH */
    return SWEEP_LIST;
)    /* end make_sweep_list */



/* Searches considered list (CL). if sweep link (SLINK) is the 2nd endpoint
   of an edge, that edge is returned to complete its processing. If no
   match is found a null pointer is returned*/

CONSIDERED_LINK *under_consideration(SLINK,CL)
            SWEEP_LINK *SLINK;
```

123

```c
            CONSIDERED_HEAD *CL;
{
            CONSIDERED_LINK *NEXT_CL=CL->LINKS;

            while (NEXT_CL) {
            if (NEXT_CL->SL1==NULL)
                        printf("\nWarning CL with no SL1");
            if (NEXT_CL->SL2==NULL)
                        printf("\nWarning CL with no SL2");
                        if (NEXT_CL->SL2==SLINK){
                                        return NEXT_CL; /* retrun ptr if in
list*/
                        }
                else
                                NEXT_CL=NEXT_CL->NEXT;
            }
            return NEXT_CL;    /* returns NULL if not in list */
}   /* end under_consideration */




/* Determine the point of intersection along CL's edge which occurs with
   the ray originating from the observer's position (X,Y,Z) along ANGLE.
   The distance to this intersection ios also calculated.

   NOTE: Intersection and distance are returned by reference in variable
         addresses INT_X, INT_Y and DIST.
                It is assumed an intersection does take place (dictated by
usage
            in algorithm).*/

void line_ray_intersection(CL,ANGLE,INT_X,INT_Y,DIST)
     CONSIDERED_LINK *CL;
     double ANGLE,*INT_X,*INT_Y,*DIST;
{
            double XX,YY;   /*values at intersection */
            double dx,dy;   /*delta values*/
            double M_LINE,M_RAY; /*slope of line and ray*/
            double B_LINE,B_RAY; /*y-intercept~*/

        dy=CL->SL2->Y-CL->SL1->Y;
            dx=CL->SL2->X-CL->SL1->X;
            if ((ANGLE==CL->SL1->THETA)&&(ANGLE==CL->SL2->THETA)) {
              if (CL->SL1->DIST<=CL->SL2->DIST) {
                        XX=CL->SL1->X;
                        YY=CL->SL1->Y;
                        *DIST=CL->SL1->DIST;
                }
                else {
/*colinear cases*/
                        XX=CL->SL2->X;
```

124

```
                         YY=CL->SL2->Y;
                         *DIST=CL->SL2->DIST;
                    }
               }
             else {
               if ((ANGLE==90.0)||(ANGLE==180.0)) {   /*ray has no slope*/
                         XX=X;
                         M_LINE=dy/dx;
                         YY=M_LINE*XX+(CL->SL1->Y-(M_LINE*CL->SL1->X));
               }
               else {
                         M_RAY=tan(ANGLE);
                         B_RAY=Y-M_RAY*X;
                         if (CL->SL1->X==CL->SL2->X) { /*line has not
slope */
                                   XX=CL->SL1->X;
                                   YY=M_RAY*XX+B_RAY;
                         }
                         else {                      /* both line and ray have
a slope */
                            M_LINE=dy/dx;
                            B_LINE=CL->SL1->Y-M_LINE*CL->SL1->X;
                            XX=(B_LINE-B_RAY)/(M_RAY-M_LINE);
                            YY=M_RAY*XX+B_RAY;
                         }  /* end else */
                 }   /* end else */
                *DIST=trunc(sqrt(pow(XX-X,2.0)+pow(YY-Y,2.0)));   /*assign
distance*/
             } /* end else */
             *INT_X=trunc(XX);    /*assign x-y coordinates of
intersection*/
             *INT_Y=trunc(YY);
       } /* end line_ray_intersection */



/* Searches currently accepted lines. If L duplicates one of these, a 1 is
   returned. Duplications will naturally occure since each vertical line is
   common to 2 edges. */

int duplicate_vert_line(L,LIST)
          LINE *L;
          LINE_HEAD *LIST;
{
          int DUP=0;
          LINE *NEXT_L=LIST->VLINE_LIST;

          while (NEXT_L) {
                    if ((L->X1==NEXT_L->X1)&&(L->Y1==NEXT_L->Y1)&&
                                 (L->Z1==NEXT_L->Z1)&&(L->Z2==NEXT_L-
>Z2))
```

```
                                        DUP=1;
                        NEXT_L=NEXT_L->NEXT;
                }
                return DUP;
}   /* end duplicate_vert_line */




void add_vert_line(CL,SL,LIST)
            CONSIDERED_LINK *CL;
            SWEEP_LINK *SL;
            LINE_HEAD *LIST;
(
            LINE *NEW_LINE=(LINE *)malloc(sizeof(LINE));
            double len;

            len=SL->DIST;
            NEW_LINE->X1=SL->X;
            NEW_LINE->Y1=SL->Y;
            NEW_LINE->MODEL_X=SL->X;
            NEW_LINE->MODEL_Y=SL->Y;
            if (CL->MIN_Z>=SL->MIN_Z)
                        NEW_LINE->Z1=tan(CL->MIN_Z)*len+Z;   /*clipped
short*/
            else
                        NEW_LINE->Z1=tan(SL->MIN_Z)*len+Z;
            NEW_LINE->X2=SL->X;
            NEW_LINE->Y2=SL->Y;
            if (CL->MAX_Z<=SL->MAX_Z)
                        NEW_LINE->Z2=tan(CL->MAX_Z)*len+Z;   /*clipped
short*/
            else
                        NEW_LINE->Z2=tan(SL->MAX_Z)*len+Z;
            NEW_LINE->NEXT=NULL;
            if (duplicate_vert_line(NEW_LINE,LIST)==0) {
                        LIST->VERT_LINES++;
                        if (LIST->VTAIL) {
                                    LIST->VTAIL->NEXT=NEW_LINE; /*add as
last vert. line*/
                                    LIST->VTAIL=NEW_LINE;
                                    }
                        else (
                                    LIST->VTAIL=NEW_LINE;   /* 1st
vertical line added */
                                    LIST->VLINE_LIST=NEW_LINE;
                        }
            }  /* end if */
            else
                        free(NEW_LINE);
```

126

```
}    /* end add_vert_line */



/* Adds only bottom edge of a considered link (CL). Lines are only accepted
   from their MIN_SWEEP angle to the current sweep angle (THETA).*/

void add_line(CL,LIST)
            CONSIDERED_LINK *CL;
            LINE_HEAD *LIST;
{
        LINE *NEW_LINE;
            double IX,IY,DIST;
                             /*DIST req fro call to intersection but value
not used*/

 /*bottom line is visible and not just a single point*/
  if ((CL->B_VISIBILITY==1)&&(my_abs(CL->MIN_SWEEP-THETA)>0.0001)) {
            NEW_LINE=(LINE *)malloc(sizeof(LINE));
            NEW_LINE->NEXT=NULL;
            LIST->LINES++;
            if (LIST->TAIL) {
                        LIST->TAIL->NEXT=NEW_LINE; /* add non-vertical
line*/
                        LIST->TAIL=NEW_LINE;
                        }
            else {
                        LIST->TAIL=NEW_LINE;   /* 1st non-vertical line
added */
                        LIST->LINE_LIST=NEW_LINE;
                }
/* find first endpoint to accept*/
            line_ray_intersection(CL,CL->MIN_SWEEP,&IX,&IY,&DIST);
            NEW_LINE->X1=IX;
            NEW_LINE->Y1=IY;
            NEW_LINE->Z1=CL->SL1->Z;
/*find second endpoint*/
            line_ray_intersection(CL,THETA,&IX,&IY,&DIST);
            NEW_LINE->X2=IX;
            NEW_LINE->Y2=IY;
            NEW_LINE->Z2=CL->SL2->Z;
    }  /* end if */
    CL->MIN_SWEEP=THETA;
}  /* end add_line */



/* This function calculates distances from the observer along the current
   THETA to each edge on the considered list. Distances do not account for
   z infromation (height), but reflect straight line distance to the
   intersection lying in the x-y plane. */
```

```
void calculate_distances(CLIST)
        CONSIDERED_HEAD *CLIST;
{
        CONSIDERED_LINK *NEXT_CL=CLIST->LINKS;
        double IX,IY;  /*pointers and values at intersection */
        double DIST; /*distance to intersection values*/

  while (NEXT_CL) {
        line_ray_intersection(NEXT_CL,THETA,&IX,&IY,&DIST);
            NEXT_CL->DIST=DIST;
            NEXT_CL=NEXT_CL->NEXT;
  }    /* end while */
}   /* end calculate_distances */




/* When a link is put on the considered list, we must determine how much of
   it is blocked from view (along the z axis) and what affect it has on
   more distant edges.

   Notice that case 2 is not accounted for since we are dealing with a
   wire frame representation.*/

void calculate_visibility_add(CLINK,CLIST,LLIST)
        CONSIDERED_LINK *CLINK;
        CONSIDERED_HEAD *CLIST;
      LINE_HEAD *LLIST;
{
        CONSIDERED_LINK *CL=CLINK->NEXT;
        int TYPE_OCCLUSION;

        if (CLINK->NEW_VISIBILITY==1) {   /*if visible it may occlude
others*/
            while (CL) {
                    if (CL->NEW_VISIBILITY==1) { /*can only block
visible lines*/
                                TYPE_OCCLUSION=occlusion(CLINK,CL);

                                switch (TYPE_OCCLUSION) {
                                case 4:
/*totally occluded*/
                                        CL->NEW_VISIBILITY=0;
                                        CL->NEW_B_VISIBILITY=0;
                                        break;
                                case 3:                /*bottom
occluded*/
                                        CL->NEW_B_VISIBILITY=0;
                                        CL->NEW_MIN_Z=CLINK-
>NEW_MAX_Z;
                                        break;
/*                              case 2:
```

```
                                                        CL->NEW_MIN_Z=CLINK-
>NEW_MAX_Z;

                                                        break;
*/                                      case 1:
/*top occluded*/

                                                        CL->NEW_MAX_Z=CLINK-
>NEW_MIN_Z;

                                                        break;
                                        ) /*end switch*/
                        )  /* end if */
                        CL=CL->NEXT;
                ) /* end while */
            ) /* end if */
)  /* end calculate_visibility_add */


/* Calculate the visibility of the vertical edge (if any) residing on the
   2nd endpoint of a link which is being passed by the sweep (thus removed
   from the considered list)*/

void calc_vis_remove(CL,CLIST)
            CONSIDERED_LINK *CL;
            CONSIDERED_HEAD *CLIST;
{
            CONSIDERED_LINK *NEXT_CL=CLIST->LINKS;
            int TYPE_OCCLUSION;

/*now calc visibility bounds of SL2's vertical line if there is one*/
    if (CL->SL2->V->VERT_EDGE) {
            while ((CL!=NEXT_CL)&&(CL->NEW_VISIBILITY==1)) {
                        if (CL->SL2->THETA==NEXT_CL->SL1->THETA)
                            TYPE_OCCLUSION=0;
                        else
                            TYPE_OCCLUSION=occlusion(NEXT_CL,CL);
                        switch (TYPE_OCCLUSION) {
                          case 4:
                                        CL->NEW_VISIBILITY=0;
                                        break;
                          case 3: case 2:
                                        CL->NEW_MIN_Z=NEXT_CL->NEW_MAX_Z;
                                        break;
                          case 1:                    /*top of B occluded*/
                                        CL->NEW_MAX_Z=NEXT_CL->NEW_MIN_Z;
                                        break;
                        )  /*end switch*/
                        NEXT_CL=NEXT_CL->NEXT;
            ) /* end while */
        CL->VISIBILITY=CL->NEW_VISIBILITY;
            CL->MIN_Z=CL->NEW_MIN_Z;
            CL->MAX_Z=CL->NEW_MAX_Z;
    ) /* end if */
```

```
    else
        CL->VISIBILITY=0;
)    /* end calc_vis_remove */


/* If visibility has been altered from last time, we must accept lines
which
   were already visible and reset the value of MIN_SWEEP to reflect where
along
   the edge these new values start.*/

int visibility_changes(CL)
         CONSIDERED_LINK *CL;
(

         int CHANGES=0;
         double EXP_MIN_Z, EXP_MAX_Z; /*expected coverage based on
perspective*/

         EXP_MIN_Z=trunc(atan((CL->SL1->Z-Z)/CL->DIST));
         EXP_MAX_Z=trunc(atan((CL->UPPER_Z-Z)/CL->DIST));

         if (CL->VISIBILITY!=CL->NEW_VISIBILITY)
                 CHANGES++;
         if (CL->B_VISIBILITY!=CL->NEW_B_VISIBILITY)
                 CHANGES++;
         if (EXP_MIN_Z!=CL->NEW_MIN_Z)
                 CHANGES++;
         if (EXP_MAX_Z!=CL->NEW_MAX_Z)
                 CHANGES++;
         return CHANGES;
)




void update_visibility(CLIST,LLIST)
         CONSIDERED_HEAD *CLIST;
         LINE_HEAD *LLIST;
(

         CONSIDERED_LINK *CL=CLIST->LINKS;

         while (CL) (
                 if (visibility_changes(CL)!=0) (
                     if ((CL->B_VISIBILITY==1)&&(IN_MAIN))
                             add_line(CL,LLIST);
                     CL->VISIBILITY=CL->NEW_VISIBILITY;
                     CL->B_VISIBILITY=CL->NEW_B_VISIBILITY;
                     CL->MIN_Z=CL->NEW_MIN_Z;
                     CL->MAX_Z=CL->NEW_MAX_Z;
                     CL->MIN_SWEEP=THETA;     /*values only affect
here on*/
                 )
```

```
                                   CL=CL->NEXT;
                  }
}


/* Visibility must be periodically recomputed to account for the effects
of
   perspective as the sweep progresses around 360 degrees.*/

void recompute_visibility(CLIST,LLIST)
                  CONSIDERED_HEAD *CLIST;
                  LINE_HEAD *LLIST;
{
                  CONSIDERED_LINK *CL=CLIST->LINKS;

                  calc_current_z_coverage(CLIST);      /*will change due to
perspective*/
                  while (CL) {                                  /*add each link again*/
                             calculate_visibility_add(CL,CLIST,LLIST);
                             CL=CL->NEXT;
                  }
                  update_visibility(CLIST,LLIST);      /*see if changes occured*/
}   /* end recompute_visibility */




/* Add a new link to the considered list (sorted by distance from observer
   in the x-y plane). If a vertical edge resides on the links first endpoint
   accept it based on the edges computed visibility*/

void add_considered_link(CL,CLIST,LLIST)
                  CONSIDERED_LINK *CL;
                  CONSIDERED_HEAD *CLIST;
             LINE_HEAD *LLIST;
{
                  CONSIDERED_LINK *NEXT_CL=CLIST->LINKS;

                  if (CLIST->LINKS) (/*recalc distances for insert*/
                    calculate_distances(CLIST);
                    if (CL->DIST<NEXT_CL->DIST) {
                              CL->NEXT=CLIST->LINKS;
                              CLIST->LINKS=CL;/*add as 1st element*/
                    }   /* end if */
                    else {
                              while ((NEXT_CL->NEXT)&&(NEXT_CL->NEXT->DIST<CL-
>DIST)) {
                                   NEXT_CL=NEXT_CL->NEXT;
                      }
/*keep ones leaning in towards camera 1st on list*/
                              while (((NEXT_CL->NEXT)&&(NEXT_CL->NEXT-
>DIST==CL->DIST))&&
```

131

```
                                        (ccw2(CL->SL1,CL->SL2,NEXT_CL->NEXT-
>SL2)))  {

                                 NEXT_CL=NEXT_CL->NEXT;
                  }
                  CL->NEXT=NEXT_CL->NEXT;
                  NEXT_CL->NEXT=CL;
            }    /* end else */
            recompute_visibility(CLIST,LLIST);
      )   /* end if */
      else {
        CLIST->LINKS=CL;/*1st element added to null list*/
        CL->VISIBILITY=1;      /*so must be visible*/
        CL->B_VISIBILITY=1;      /*so must be visible*/
      }
      if ((IN_MAIN)&&(((CL->VISIBILITY==1)&&(CL->MIN_Z<CL->MAX_Z))
          &&(CL->SL1->V->VERT_EDGE)))
          add_vert_line(CL,CL->SL1,LLIST);
}   /* end add_considered_link */




/* Remove a CL from the list*/

void remove_cl(CL,CLIST)
            CONSIDERED_LINK *CL;
            CONSIDERED_HEAD *CLIST;
{
            CONSIDERED_LINK *NEXT_CL=CLIST->LINKS;

            if (CL==NEXT_CL) {  /* removing 1st link */
              CLIST->LINKS=NEXT_CL->NEXT;
              free(CL);    /*deallocate memory*/
            }
            else {
              while ((NEXT_CL->NEXT)&&(NEXT_CL->NEXT!=CL)) {
                      NEXT_CL=NEXT_CL->NEXT;
              }
              if (NEXT_CL->NEXT) {
                      NEXT_CL->NEXT=CL->NEXT;
                      free(CL);    /*deallocate memory*/
              }
            }  /* end else */
}   /* end remove_cl */




/* The sweep has progresses to the end of link CL. We need to inspect the
   visibility and accept both the bottom edge and vertical line (at 2nd
   endpoint) if required.
      Once this is done, visibility of the entire considered list (CLIST)
```

132

```
        must be recomputed to account for perspective and the deleted edge*/

void complete_line(CL,CLIST,LLIST)
            CONSIDERED_LINK *CL;
            CONSIDERED_HEAD *CLIST;
            LINE_HEAD *LLIST;
{
            LINE *L;

            if ((CL->VISIBILITY==1)&&(CL->B_VISIBILITY==1))
              add_line(CL,LLIST);   /*also checks for and adds right vert
line*/
            calculate_distances(CLIST);
            calc_current_z_coverage(CLIST);
            calc_vis_remove(CL,CLIST);
            if ((CL->SL2->V->VERT_EDGE)&&(CL->VISIBILITY==1))
               add_vert_line(CL,CL->SL2,LLIST);
            remove_cl(CL,CLIST); /*if not visible no changes needed
before removal*/
            recompute_visibility(CLIST,LLIST);
}    /* end complete_line */




/* These occlusion codes applie if both links begin at the same vertex
   in the model.*/

int overlay_occlusion(F,B)
            CONSIDERED_LINK *F, *B;
{
            int TYPE=0; /*default is no occlusion occurs*/

  if (F->NEW_MIN_Z<=B->NEW_MIN_Z) {
            if (F->NEW_MAX_Z>=B->NEW_MAX_Z)
                      TYPE=4;    /* totally occluded*/
            else {
                      if (F->NEW_MAX_Z>=B->NEW_MIN_Z)
                                 TYPE=3;  /*bottom of B occluded*/
            }
  } /* end if */
  else {
            if (F->NEW_MAX_Z<B->NEW_MAX_Z)
                      TYPE=2;    /*middle prtion of B occluded*/
            else {
                      if (F->NEW_MIN_Z<=B->NEW_MAX_Z)
                                 TYPE=1;  /*top of B occluded*/
            }
  } /* end else */
  /* otherwise there is no occlusion */
  return TYPE;
}    /* end overlay_occlusion */
```

```c
/* The type of occlusion imposed upon the back edge (B) by the front edge
(F)
   is determined: return value is 0,1,2,3, or 4 */

int occlusion(F,B)
            CONSIDERED_LINK *F, *B;
{
            int TYPE=0; /*default is no occlusion occurs*/

/*No occlusion if edges fall on the same plane or are end-to-end*/
if (((F->SL1->THETA==B->SL2->THETA)||(F->SL2->THETA==B->SL1->THETA))||
      ((F->MIN_Z==F->MAX_Z)||(colinear(F,B))))
    TYPE=0;
else {
    if (((F->SL1->DIST==B->SL1->DIST)&&(F->SL1->THETA==B->SL1->THETA))
            &&(B->UPPER_Z<9999.0))
            TYPE=overlay_occlusion(F,B);
    else {
      if (F->NEW_MIN_Z<B->NEW_MIN_Z) {
            if (F->NEW_MAX_Z>B->NEW_MAX_Z)
                    TYPE=4;     /* totally occluded*/
            else {
                    if (F->NEW_MAX_Z>B->NEW_MIN_Z)
                                TYPE=3;  /*bottom of B occluded*/
            }
      } /* end if */
      else {
            if (F->NEW_MAX_Z<B->NEW_MAX_Z)
                    TYPE=2;     /*middle prtion of B occluded*/
            else {
                    if (F->NEW_MIN_Z<B->NEW_MAX_Z)
                                TYPE=1;  /*top of B occluded*/
            }
      } /* end else */
    } /* end else */
 } /* end else */
  /* otherwise there is no occlusion */
  return TYPE;
}   /* end occlusion */


/* This is the primary function which will be called from outside this
file.
A list of sweep links is constructed based on the model (W) and the
observer's
position (EYE_X,EYE_Y,EYE_Z).
   Next edges straddling 0 degrees are placed on the considered list (if
they
are ccw). Then main processing begins and each sweep link and its
```

predicessor
pair is inspected. If the circuit from observer to SL to prev(SL) is ccw,
the
SL's are put into a considered link (CL) and added to the considered list
(CLIST).
  As the sweep progresses throught the sweep links: visibility is updated,
lines are accepted, and edges are removed from CLIST (as they are passed).

        OUTPUT: LINE_LIST   structure pointing to 2 list of lines
                                      (vertical and non-vertical accepted lines)

*/


```
LINE_HEAD *conduct_visibility_sweep(W,EYE_X,EYE_Y,EYE_Z)
           WORLD *W;
           double EYE_X,EYE_Y,EYE_Z;
{
           SWEEP_LINK *NEXT_SL, *SWEEP_LIST=NULL;
           CONSIDERED_LINK *CL, *PAST_CL;
           CONSIDERED_HEAD *CLIST=make_considered_head();
           LINE_HEAD *LINE_LIST=make_line_head();
           int STRADDLERS=0;

           IN_MAIN=0;     /*still processing straddlers*/
           X=EYE_X;
           Y=EYE_Y;
           Z=EYE_Z;
           SWEEP_LIST=make_sweep_list(W);
           NEXT_SL=SWEEP_LIST;
/* Add all visible straddlers*/
           while (NEXT_SL) {
                       THETA=NEXT_SL->THETA;
                       if ((ccw(NEXT_SL,NEXT_SL->PREV)==1)&&
                           (NEXT_SL->THETA>NEXT_SL->PREV->THETA)) {
                                        CL=make_considered_link(NEXT_SL);

add_considered_link(CL,CLIST,LINE_LIST);
                                        CL->MIN_SWEEP=0.0;
                                        STRADDLERS=1;
                       }
                       NEXT_SL=NEXT_SL->NEXT;
           }  /* end while */
           NEXT_SL=SWEEP_LIST;
           THETA=0.0;
           IN_MAIN=1;
/* Process all of sweep list*/
           while (NEXT_SL) {
                       THETA=NEXT_SL->THETA;
                       while
(PAST_CL=under_consideration(NEXT_SL,CLIST)) {
```

```
complete_line(PAST_CL,CLIST,LINE_LIST);
                    }
                    if (ccw(NEXT_SL,NEXT_SL->PREV)==1) {
                                CL=make_considered_link(NEXT_SL);

add_considered_link(CL,CLIST,LINE_LIST);
                    }
                    NEXT_SL=NEXT_SL->NEXT;
          }   /* end while */
          if (STRADDLERS) {    /* have lines crossing ZERO degrees */
                    THETA=0.0;
                    calculate_distances(CLIST);
                    CL=CLIST->LINKS;
                    while (CL) {
                                if ((CL->VISIBILITY==1)&&(CL-
>B_VISIBILITY==1))
                                        add_line(CL,LINE_LIST);
                                CL=CL->NEXT;
                    }
    }   /* end if */
    free_clist(CLIST);
    free_sweep_list(SWEEP_LIST);
    return LINE_LIST;
}   /* end conduct_visibility_sweep */
```

```
/
*************************************************************************
*****

  FILENAME: 2d+.h
  AUTHOR:   LT James Stein
  CONTENTS: 2d+ model support tools (for building, displaying, searching,
            and deallocating a model)
  DATE:     Mar 1992
  COMMENTS: A 'world' consists of a list of polyhedrons (PH) Each PH is in
turn a list of polygons (PG). Each PG is a list of VERTICIES which contain
the X,Y, and Z coordinates of that point in the world.
  File 5th.h is an example construction file which uses these functions to
build a model of the 1st half of Spanagel Hall's 5th floor.


*************************************************************************
*****/


/*  constants */
#define PI              3.141592653589793
#define MAX_LEN         30

/* typedefs: Define structures to be used for representing a 3-d world */

typedef struct vertex {
  float X,Y;
  struct vertex
    *NEXT, *PREV,
    *VERT_EDGE;
) VERTEX;

/* WHERE:  VERT_EDGE = pointer to upper vertex of vertical edge

------------------------------------*/

typedef struct poly_link {
  struct polygon *REF_POLY;
  struct poly_link *NEXT, *PREV;
) POLY_LINK;

/*----------------------------------*/

typedef struct polygon {
  int DEGREE, C_DEGREE, FLOOR, CONVEX;
  float Z_VALUE;
  VERTEX *VERTEX_LIST;
  POLY_LINK *CEILING_LIST;
  struct polygon
    *NEXT, *PREV;
) POLYGON;
```

```
/* WHERE:    DEGREE = # of vertices
                        FLOOR, CONVEX = booleans
                        Z_VALUE = local Z position poly located at
                        CEILING_LIST, FLOOR_LIST = list of associated
poly's


------------------------------------*/

typedef struct instance {
  char NAME[MAX_LEN];
  float X, Y, Z, ROTATION,
        PIVOT_X, PIVOT_Y;
  struct instance *NEXT, *PREV;
} INSTANCE;


/* WHERE:    NAME = something like 'rm501'
                        X, Y, Z = position to instantiate PH into world
                        ROTATION = degrees to rot about Z axis

------------------------------------*/

typedef struct polyhedron {
  char CLASS[MAX_LEN];
  int DEGREE, I_DEGREE, OBSTACLE, FIXED;
  POLYGON *POLYGON_LIST;    /*ordered by Z value*/
  INSTANCE *INSTANCE_LIST; /*ordered by Z value*/
  struct polyhedron *NEXT, *PREV;
} POLYHEDRON;

/* WHERE:    CLASS = general name like 'door'
                        DEGREE = # of polygons
                        OBSTACLE and FIXED = booleans
                        CEILING_LIST, FLOOR_LIST = list comprise all
polygons
                        INSTANCE_LIST = all tranformations of object into
world

------------------------------------*/

typedef struct world {
  char NAME[MAX_LEN];
  int DEGREE;
  POLYHEDRON *POLYHEDRON_LIST;
} WORLD;

/* WHERE:    NAME = label for world
                        DEGREE = number of object representations
                        POLYHEDRON_LIST points to them
```

```
  */



  /********************************************************************

  The following routines are called to allocate memory for a structure
  (WORLD, POLYHEDRON, POLYGON, or VERTEX). Pointers are initialized to NULL
  and the DEGREE field is set to 0;

  ********************************************************************/

  WORLD *create_world()
  {
    WORLD *W;
    int i;

    /* allocate memory for a world */
    if((W = (WORLD *)malloc(sizeof(WORLD))) == NULL) {

      printf("\ncannot create a world\n");
    }
    /* initialize fields */
    W->DEGREE = 0;
    W->POLYHEDRON_LIST = NULL;
    for (i=0; i<MAX_LEN; ++i) {
      W->NAME[i]=' ';
    }
    return(W);
  }



  /*----------------------------------------------------------------*/



  POLYGON *create_polygon()
  {
    POLYGON *P;

    /* allocate memory for a polygon */
    if((P = (POLYGON *)malloc(sizeof(POLYGON))) == NULL) {
      printf("\cannot create a polygon");
    }

    /* initialize fields */
    P->DEGREE = 0;
    P->Z_VALUE= 0.0;
    P->VERTEX_LIST = NULL;
    P->CEILING_LIST= NULL;
    P->NEXT = NULL;
    P->PREV = NULL;
```

139

```c
  return(P);
}


INSTANCE *create_instance()
{
  INSTANCE *I;
  int i;

  I= (INSTANCE *)malloc(sizeof(INSTANCE));
  for (i=0; i<MAX_LEN; ++i) {
    I->NAME[i]=' ';
  }
  I->NEXT = NULL;
  I->PREV = NULL;
  return I;
}


/*------------------------------------------------------------------*/


POLY_LINK *create_poly_link() {

  POLY_LINK *P;
  P=(POLY_LINK *)malloc(sizeof(POLY_LINK));
  P->REF_POLY = NULL;
  P->NEXT = NULL;
  P->PREV = NULL;
  return P;
}


/*------------------------------------------------------------------*/

POLYHEDRON *create_polyhedron()
{
  POLYHEDRON *P;
  int i;

  P=(POLYHEDRON *)malloc(sizeof(POLYHEDRON));
  for (i=0; i<MAX_LEN; ++i) {
    P->CLASS[i]=' ';
  }
  P->DEGREE=0;
  P->POLYGON_LIST=NULL;
  P->NEXT=NULL;
  P->PREV=NULL;
  P->INSTANCE_LIST=NULL;
  return P;
}  /* end create_polyhedron */
```

```
/*------------------------------------------------------------*/


VERTEX *create_vertex()
{
  VERTEX *V;

  V=(VERTEX *)malloc(sizeof(VERTEX));
  V->NEXT=NULL;
  V->PREV = NULL;
  V-> VERT_EDGE =NULL;
  return V;
}


/********************************************************************

  The following routines are used for memory deallocation. Each type of
list is stepped through to free it's component structures. Higher level
structures call the free routine for the next lower level to deallocate
side lists (i.e. free_world calls free_polyhedron).

********************************************************************/


void free_pg(PG)
     POLYGON *PG;
{
  VERTEX *NEXT_V, *TRASH;
  POLY_LINK *NEXT_LINK, *TRASH2;

  NEXT_V=PG->VERTEX_LIST;    /*free vertex list*/
  while (NEXT_V) {
    TRASH=NEXT_V;
    NEXT_V=NEXT_V->NEXT;
    free(TRASH);
  }
  NEXT_LINK=PG->CEILING_LIST;
  while (NEXT_LINK) {            /*free links used to reference ceilings*/
    TRASH2=NEXT_LINK;
    NEXT_LINK=NEXT_LINK->NEXT;
    free(TRASH2);
  }
  free(PG);  /*free parent polygon structure */
}   /* end free_pg */


/*------------------------------------------------------------*/
```

```
void free_ph(PH)
     POLYHEDRON *PH;
{
  POLYGON *NEXT_PG, *TRASH;
  INSTANCE *NEXT_I, *TRASH2;

  NEXT_PG=PH->POLYGON_LIST;
  while (NEXT_PG) {            /*free the list of polygons*/
    TRASH=NEXT_PG;
    NEXT_PG=NEXT_PG->NEXT;
    free_pg(TRASH);
  }
  NEXT_I = PH->INSTANCE_LIST;
  while(NEXT_I) {              /*free the list of instances*/
    TRASH2= NEXT_I;
    NEXT_I= NEXT_I->NEXT;
    free(TRASH2);
  }
  free(PH);   /* release parent structure */
}   /* end free_ph */


/*------------------------------------------------------------------*/

void free_world(W)
     WORLD *W;
{
  POLYHEDRON *NEXT_PH, *TRASH;

  if (W) {
    NEXT_PH=W->POLYHEDRON_LIST;
    while (NEXT_PH) {              /*free the list of polyhedra*/
      TRASH=NEXT_PH;
      NEXT_PH=NEXT_PH->NEXT;
      free_ph(TRASH);
    }
  }
  free(W);
}   /* end free_world */


/*********************************************************************

  The next group of functions is used to display the world. A single
polygon, a single polyhedron, or the entire world can be displayed.
Display is in text format to the standard output device.

*********************************************************************/
```

```c
void display_pg(PG)
     POLYGON *PG;
{
  POLYGON *NEXT_PG;
  POLY_LINK *NEXT_C;
  VERTEX *NEXT_V;
  int V_NUM=1, PRINTED=0;

  printf("\nDEGREE: %d FLOOR: %d  Convex: %d ",PG->DEGREE,PG->FLOOR,
              PG->CONVEX);
  printf("\nZ = %.2f:\n",PG->Z_VALUE);
  NEXT_V=PG->VERTEX_LIST;
  while (NEXT_V) {
    if (PRINTED>3) {        /* three vertices per line*/
      printf("\nV#%d(%.2f,%.2f) ",V_NUM,NEXT_V->X,NEXT_V->Y);
      PRINTED=1;
    }
    else {
      printf("V#%d(%.2f,%.2f) ",V_NUM,NEXT_V->X,NEXT_V->Y);
      PRINTED++;
    }
    NEXT_V=NEXT_V->NEXT;
    V_NUM++;
  } /*end while */
  if (PG->FLOOR==1)
    printf("\nAssociated ceilings (%d): ",PG->C_DEGREE);
  NEXT_C= PG->CEILING_LIST;
}    /* end display_pg */


/*------------------------------------------------------------------*/


void display_ph(PH)
     POLYHEDRON *PH;
{
  POLYGON *NEXT_PG;
  int PG_NUM,F_CNT=1,C_CNT=1,I_CNT=1;
  char dummy;
  INSTANCE *NEXT_I;

  printf("\nPOLYHEDRON (%s):\n Obstacle: %d  Fixed: %d \n",
              PH->CLASS,PH->OBSTACLE,PH->FIXED);
  printf("\nComponent polygons (%d):\n ",PH->DEGREE);
  NEXT_PG=PH->POLYGON_LIST;
  printf("\n\nList of floors:");
  while (NEXT_PG) {
    if (NEXT_PG->FLOOR==1) {
      printf("\n\nFLOOR# %d ",F_CNT);
      display_pg(NEXT_PG);                /*display floor polygons*/
      F_CNT++;
```

143

```c
      ) /* end if */
      NEXT_PG=NEXT_PG->NEXT;
    ) /* end while */
    NEXT_PG=PH->POLYGON_LIST;
    printf("\n\nList of ceilings:");
    while (NEXT_PG) {
      if (NEXT_PG->FLOOR==0) {
        printf("\n\nCEILING # %d ",C_CNT);
        display_pg(NEXT_PG);                    /*display ceilings*/
        C_CNT++;
      ) /* end if */
      NEXT_PG=NEXT_PG->NEXT;
    ) /* end while */
    printf("\n\nThe following instantiations of this polyhedron exist:");
    fflush(stdout);
    if (PH==NULL) {
              printf("\n\ndereferencing null pointer in display_ph\n\n");
              fflush(stdout);
    )
    NEXT_I=PH->INSTANCE_LIST;
    while(NEXT_I) {
      printf("\n\nInstance #%d (%s): ",I_CNT,NEXT_I->NAME);
      fflush(stdout);
      printf("\nLocation: (%.2f,%.2f,%.2f)",NEXT_I->X,NEXT_I->Y,NEXT_I->Z);
      fflush(stdout);
      printf("Rotated: %.2f degrees about point: (%.2f,%.2f)\n",
                NEXT_I->ROTATION,NEXT_I->PIVOT_X,NEXT_I->PIVOT_Y);
      fflush(stdout);
      I_CNT++;
      NEXT_I=NEXT_I->NEXT;
    ) /* end while */
)    /* end display_ph */




/*--------------------------------------------------------------------*/


void display_world(W)
     WORLD *W;
(
  POLYHEDRON *PH;
  POLYGON *PG;
  int NUM_PH=1;

if (W) {
  printf("\nWorld Name: %s",W->NAME);
  printf("\n\nWorld has:\n %d POLYHEDRONS\n ",W->DEGREE);
  PH=W->POLYHEDRON_LIST;
  while (PH) {
    printf("\n\nPH #%d \n",NUM_PH);
```

```
      NUM_PH++;
      display_ph(PH);
      PH=PH->NEXT;
   }
}  /* end if */
}  /* end display world */


/*******************************************************************

   The following functions are used by the construction file
   to add structures (i.e.- POLYHEDRON, POLYGON, VERTEX, and INSTANCE)
   and associations (i.e.- vertical edges and floor->ceiling associations)
   to a world.

********************************************************************/

void add_edge(V1,V2)
      VERTEX *V1, *V2;   /*lower and upper vertices of edge*/
{
  if (V1->VERT_EDGE)
    printf("\nWarning reassignment of vertical edge attempted!!!");
  else
    V1->VERT_EDGE = V2;
} /* end add_edge */



/*******************************************************************/

void add_ceiling(PG,C)
      POLYGON *PG, *C;    /*floor and its new ceiling*/
{
  POLY_LINK *NEW_C,*NEXT_C;
  int FOUND=0;

  if (PG->CEILING_LIST) {
    NEXT_C= PG->CEILING_LIST;
    if (NEXT_C->REF_POLY==C)
      FOUND=1;
    else
      while (NEXT_C->NEXT) {
             if (NEXT_C->NEXT->REF_POLY==C)
               FOUND=1;
             NEXT_C=NEXT_C->NEXT;
      } /* end while */
    if (FOUND==0) {
      NEW_C=create_poly_link();  /*link onto end of list*/
      NEW_C->REF_POLY=C;
      NEW_C->PREV=NEXT_C;
      NEXT_C->NEXT=NEW_C;
      PG->C_DEGREE++;
    } /* end if */
```

145

```
        else
          printf("\nWarning - attempted to add ceiling which exists");
    } /* end if */
    else {
      NEW_C=create_poly_link(); /*adding 1st ceiling to list*/
      NEW_C->REF_POLY=C;
      PG->CEILING_LIST=NEW_C;
      PG->C_DEGREE++;
    } /* end else */
} /* end add_ceiling */



/*******************************************************************


  X,Y,Z is the position in the parent world at which the pivot point
  is to be placed.
  PIVOT_X and PIVOT_Y specify th local coordinates (in POLYHEDRON) of
  the objects pivot point.
  ROT is the number of degrees the object should be rotated about this
  pivot point.

  ****************************************************************/


void *add_instance(NAME,LEN,PH,X,Y,Z,PIVOT_X,PIVOT_Y,ROT)
     POLYHEDRON *PH;
     float X,Y,Z,PIVOT_X,PIVOT_Y,ROT;
     char NAME[];    /*label for instance and number of characters in
label*/
     int LEN;
{
  INSTANCE *I,*TEMP_I,*NEXT_I;
  int i;

  I=create_instance();   /*allocate and initialize memory*/
  for (i=0;i<=LEN;++i) {
    I->NAME[i]=NAME[i];
  }
  I->X=X;
  I->Y=Y;
  I->Z=Z;
  I->PIVOT_X=PIVOT_X;
  I->PIVOT_Y=PIVOT_Y;
  I->ROTATION=ROT;
  /*order by z*/
  if (PH->INSTANCE_LIST==NULL) {
    PH->INSTANCE_LIST=I;
  }
  else {
    NEXT_I=PH->INSTANCE_LIST;
    if (Z<=NEXT_I->Z) {
      I->NEXT=NEXT_I;
```

```
              NEXT_I->PREV=I;        /* add to head of list*/
              PH->INSTANCE_LIST=I;
          } /* end if */
          else {
            while (NEXT_I->NEXT&&NEXT_I->NEXT->Z<Z) {
                  NEXT_I=NEXT_I->NEXT;  /*scan to insertion point*/
            }
            if (NEXT_I->NEXT) {
                  I->NEXT=NEXT_I->NEXT; /*add to middle of list*/
                  I->PREV=NEXT_I;
                  NEXT_I->NEXT=I;
                  I->NEXT->PREV=I;
            } /* end if */
            else {
                  I->PREV=NEXT_I;     /*add as last instance*/
                  NEXT_I->NEXT=I;
            } /* end else */
          } /* end else */
      } /* end else */
      PH->I_DEGREE++;              /*keep track of the number of instances*/
} /* end add_instance */


/***********************************************************************

  The remaining add functions create and add structures to the world.
  Pointers to each newly added structure are returned to the caller for
  future use.

  ***********************************************************************/


VERTEX *add_vertex(PG,X,Y)
      POLYGON *PG;      /* parent polygon to add vertex to*/
      float X,Y;        /*local coordinates of vertex*/
{
  VERTEX *V, *NEXT_V;

  V=create_vertex();
  V->X=X;
  V->Y=Y;
  if (PG->VERTEX_LIST==NULL)
    PG->VERTEX_LIST=V;
  else {
    NEXT_V=PG->VERTEX_LIST;
    while (NEXT_V->NEXT) {
      NEXT_V=NEXT_V->NEXT;      /* scan to end of list */
    }
    NEXT_V->NEXT=V;   /* add to end of list to retain order added*/
    V->PREV=NEXT_V;
  } /* end else */
```

```
    PG->DEGREE++;
    return V;
) /* end add_vertex */



/*------------------------------------------------------------*/



POLYGON *add_pg(PH,Z,FLOOR,CONVEX)
    POLYHEDRON *PH;              /*parent structure*/
    float Z;                     /*height in local coordinates*/
    int FLOOR,CONVEX;            /*boolean values*/
(
    POLYGON *PG,*NEXT_PG;

  PG=create_polygon();
  PG->Z_VALUE=Z;
  PG->FLOOR=FLOOR;
  PG->CONVEX=CONVEX;
  if (PH->POLYGON_LIST==NULL)    /*sorted by Z height*/
    PH->POLYGON_LIST=PG;
  else (
    NEXT_PG=PH->POLYGON_LIST;
    if (Z<NEXT_PG->Z_VALUE) (    /*put at head of list*/
      NEXT_PG->PREV=PG;
      PG->NEXT=NEXT_PG;
      PH->POLYGON_LIST=PG;
    ) /* end if */
    else (
      while ((NEXT_PG->NEXT)&&(NEXT_PG->NEXT->Z_VALUE>Z)) (
            NEXT_PG=NEXT_PG->NEXT;
      )
      if (NEXT_PG->NEXT) (
            PG->NEXT=NEXT_PG->NEXT;    /* put in middle of list */
            PG->PREV=NEXT_PG;
            NEXT_PG->NEXT=PG;
            PG->NEXT->PREV=PG;
      ) /* end if */
      else (
            NEXT_PG->NEXT=PG;    /* put at end of list */
            PG->PREV=NEXT_PG;
      ) /* end else */
    ) /* end else */
  ) /* end else */
  PH->DEGREE++;
  return PG;
)   /* end add_pg */



/*------------------------------------------------------------*/
```

```
POLYHEDRON *add_ph(CLASS,LEN,W,FIXED,OBSTACLE)
     char CLASS[];      /*class name*/
     WORLD *W;              /*world to add polyhedron to*/
     int FIXED,OBSTACLE,LEN;   /* 2 booleans and the length of CLASS*/
{
  POLYHEDRON *PH,*NEXT_PH;
  int i;

  PH=create_polyhedron();
  for (i=0;i<=LEN;++i) {
    PH->CLASS[i]=CLASS[i];
  }
  PH->FIXED=FIXED;
  PH->OBSTACLE=OBSTACLE;
  if (W->POLYHEDRON_LIST==NULL)
    W->POLYHEDRON_LIST=PH;
  else {
    NEXT_PH=W->POLYHEDRON_LIST;
    while (NEXT_PH->NEXT) {
      NEXT_PH=NEXT_PH->NEXT;     /*scan to end of list*/
    }
    NEXT_PH->NEXT=PH;
    PH->PREV=NEXT_PH;
  } /* end else */
  W->DEGREE++;
  return PH;
}  /* end add_ph */




/*------------------------------------------------------------------*/


WORLD *add_world(NAME,LEN)
     char NAME[];  /*label and its length*/
     int LEN;
{
  WORLD *W;
  int i;

  W=create_world();
  for (i=0;i<LEN;++i) {  /*assign label*/
    W->NAME[i]=NAME[i];
  }
  return W;
} /* end add_world */



/*******************************************************************
```

```
       find_ph will find and display a polyhedron based on its class
       name. Component polygons and instances will be listed to the screen.
       If the pointer to a polyhedron is needed: change this function
       return PH.

    ********************************************************************/

void find_ph(LABEL,W)
     char LABEL[MAX_LEN];      /*class label to look for*/
     WORLD *W;                 /*world to search*/
{
  POLYHEDRON *NEXT_PH, *PH;
  int FOUND=0, i, MATCH;

  if (W) {
    printf("\nsearching for label: (");
    for (i=0;i<MAX_LEN;++i) {
            printf("%c",LABEL[i]);
    }
    printf(")\n");
    NEXT_PH=W->POLYHEDRON_LIST;
    while (NEXT_PH) {
      MATCH=1;
      for (i=0;i<MAX_LEN;++i) {
            if (NEXT_PH->CLASS[i]!=LABEL[i]){
              MATCH=0;          /*at least one character is different*/
            }
      }
      if (MATCH==1) {
            FOUND++;
            PH=NEXT_PH;
      }
      NEXT_PH=NEXT_PH->NEXT;
    } /* end while */
    if (FOUND==0)
      printf("\nNo polyherdon found under this label!\n");
    else {
      display_ph(PH);      /*show the polygon found*/
      if (FOUND>1)
            printf("\nWarning non-unique label (last occurance
listed).\n");
    } /* end else */
  } /* end if */
  else
    printf("\n\nCannot find polyhedron since world is empty !!!\n");
} /* end find_ph */
```

150

APPENDIX B - IMAGE UNDERSTANDING ROUTINES

The following routines implement the anlaysis of video images, localization of objects in those images, evaluation of the objects, and determination of avoidance parameters based on this analysis. The files included are the following:
locatetypes.h locateio.h locateimagesupport.h locateobjectsupport.h

```
/*------------------------------------------------------------------
      FUNCTION: locatetypes.h
       PURPOSE: Defines structures used in locate.c
       AUTHORS: Kevin Peterson & Mark DeClue
          DATE: 24 Mar 93

    STRUCTURES: CMAPIMAGE, NPSIMAGE, EDGE, POINT, POSE, IMG_LINE,
               OBJECT_DATA
       GLOBALS: none
      COMMENTS: None
    ------------------------------------------------------------*/


#define RGBA 1        /* RGBA 24 bit images (alpha is 0xff filled) */

#define CMAPPED 2      /* color mapped images */

#define RGBAWITHALPHA 3  /* RGBA 32 bit images where alpha is read/saved
                 in the image files. */


/* define a structure type for color mapped images */

struct cmapimage
{

  short *bitsptr;  /* the bits for the short images */

  long nentries;   /* the total number of entries in the color map */

  short *reds;     /* ptr to the red entries of the color map */

  short *greens;   /* ptr to the green entries in the color map */

  short *blues;    /* ptr to the blue entries in the color map */

  long cmapoffset; /* color map offset. i.e. the first color
            we will us in the color map */

};
typedef struct cmapimage CMAPIMAGE;  /* define a CMAPIMAGE type */
```

151

```c
/* define a union so that the top level image structure's
   last pointer can point to several different kinds of images. */

union imagedptr
{
  long *bitsptr;   /* long images need no more data than a ptr
                to the bits.              */

  CMAPIMAGE *cmapptr;  /* a color mapped image must have
                the bits and a color map so we
                need a complete structure.     */
};


/* define the top level structure for the image */

struct image
{
  long type;    /* image type */

  long xsize;  /* xsize of the image */

  long ysize;  /* ysize of the image */

  char *name;  /* ptr to string naming the image */

  union imagedptr imgdata;  /* ptrs to data for this type of image */

};
typedef struct image NPSIMAGE;   /* define an NPSIMAGE type */


typedef struct edge_region_type
{
  char name;       /* for use with "view.text" */

  int active;     /* boolean if past region is appended to
                                           a present region */

  long first_pixel; /* first and last pixels added to region */

  long last_pixel;

  long xmin;       /* min & max pixels for previous row */

  long xmax;


long pres_xmin, pres_xmax, prev_xmin, prev_xmax; /* for ali1.c */
double min_phi, max_phi;
```

152

```c
    int pri_phi_area;  /* for use with constant segmented angles */
    float sec_phi_area;
    double temp_phi;


    double avg_phi;  /* for use with dynamic averaging phi */

    double sum_phi;

    /* Least Squares Fit momments: */
    long   m00;      /* Number of pixels */

    double m10;      /* Sum x */

    double m01;      /* Sum y */

    double m11;      /* Sum x*y */

    double m20;      /* Sum x*x */

    double m02;      /* Sum y*y */

    struct edge_region_type *next; /* ptr to the next EDGE */

} EDGE;



struct point_type
{
  double x,y; /* x,y coordinates of the pixel endpoints */
};
typedef struct point_type POINT;


struct pose_type
{
  float x,y,theta;
};
typedef struct pose_type POSE;



typedef struct line_type
{
  char name[3]; /* for troubleshooting */

  POINT p1, p2; /* the 2 endpoints for the line */
```

```c
/* Least Squares Fit momments: */
long   m00;   /* Number of pixels */

double m10;   /* Sum x */

double m01;   /* Sum y */

double m11;   /* Sum x*y */

double m20;   /* Sum x*x */

double m02;   /* Sum y*y */

double phi;   /* Calculated normal orientation of IMG_LINE */

double dmajor; /* Length of major axis of equivallent ellipse */

double dminor; /* Length of minor axis of equivallent ellipse */

double rho;   /* Ratio dminor/dmajor */

int orient,   /* Line orientation: 1=vert, 2=horz, 3=diag */
    obstacle;  /* True if any part of line <250 pixels since may
                  be obstacle to robot */

struct line_type *next; /* ptr to the next IMG_LINE in the image */

} IMG_LINE;


/* Structure which holds the information about an object */
typedef struct object_data
{
  double range, left, right, top, bottom, width, height;
} OBJECT_DATA;
```

```
/*----------------------------------------------------------------
    FILENAME: locateio.h
     PURPOSE: Provides routines for screen and file io necessary in the
              locate program.
      AUTHOR: Mark DeClue
        DATE: 24 Mar 93

       FILES: write_lines, draw_lines, display_all, display_all_loop,
              dimensionout
     GLOBALS: none
    COMMENTS: none
    ----------------------------------------------------------------*/




/*----------------------------------------------------------------
    FUNCTION: write_lines(x,y,imgname,funcname,filesname)
     PURPOSE: Writes a list of lines with various parameters to text file.

  PARAMETERS: x - horizontal dimension of image
              y - vertical dimension of image
              imgname - name of image from which lines were generated
              funcname - name of function which generated the lines
              filesname - name of file to which lines are to be written
     RETURNS: none
   CALLLED BY: locate.c
       CALLS: fopen, fclose
    COMMENTS: none
    ----------------------------------------------------------------*/

write_lines(x,y,imgname,funcname,filesname)
   long x,y;
   char imgname[];
   char *funcname;
   char *filesname;
{
 IMG_LINE *l = Line_list_head;
 FILE *lines_file;

 lines_file = fopen(filesname,"w");
 fprintf(lines_file,"Lines for image: %s\n",imgname);
 fprintf(lines_file,"   Generated by: %s\n",funcname);
 fprintf(lines_file,"\nImage size: nr pixels x axis = %d, nr pixels y axis =  %d\n",x,y);
 fprintf(lines_file,"Extracted line segments listed in order by length.\n\n");

 while(l!=NULL)
  {
```

155

```
      fprintf(lines_file,"%s> length = %.4f, orient = %d, orientation = %.4f, m00 = %d\n",
                      l->name, l->dmajor, l->orient, l->phi, l->m00);

      fprintf(lines_file,"endpoints: (%.2f %.2f)  (%.2f %.2f)\n\n",
                      l->p1.x,l->p1.y,l->p2.x,l->p2.y);
      l = l->next;
      }
  fclose(lines_file);

} /* End write_lines */




/*----------------------------------------------------------------
      FUNCTION: draw_lines(l,color)
        PURPOSE: Draws extracted lines pointed to by l over the current image
                    in an IRIS window.
  PARAMETERS: l - pointer to list of lines to be displayed
                    color - desired color of lines
        RETURNS: none
      CALLED BY: display_all_loop
          CALLS: c3f, cmov2, move2, draw2, swapbuffers (all IRIS routines)
      COMMENTS: none
------------------------------------------------------------*/
draw_lines(l,color)
    IMG_LINE *l;
      int color;
{
  static float white[3] = {1.0,1.0,1.0}; /* rgb white */
  static float black[3] = {0.0,0.0,0.0}; /* rgb black */
  int howto;

  if(color == Black)
    c3f(black);
  else c3f(white);

  while(l!=NULL)
  {
  /*cmov2(l->p1.x,l->p1.y);  charstr(l->name); */
  /* NOTE: With comments removed, the above statements would generate
        line numbers with the line segments */

    move2(l->p1.x,l->p1.y);  draw2(l->p2.x,l->p2.y);
    l = l->next;
  }

  swapbuffers();

} /* End draw_lines */
```

156

```
/*------------------------------------------------------------
    FUNCTION: display_all(img1,img2,img3)
      PURPOSE: Displays the gradient image with or without lines, lines
                alone, grayscale image or grayscale object only.
   PARAMETERS: img1 - ptr to grayscale image
                img2 - ptr to gradient image
                img3 - ptr to gradient object only image
      RETURNS: none
    CALLED BY: locate.c
        CALLS: prefsize(x,y), winopen("name"), RGBmode(), singlebuffer()
                gconfig(), qdevice(device name), display_all_loop(i1,id1,
                i2,id2,i3), free(m), winclose("name"),
    COMMENTS: none
-------------------------------------------------------------*/


display_all(img1,img2,img3)
    NPSIMAGE *img1, /* Grayscale image */
        *img2, /* Gradient image */
        *img3; /* Gradient object only image */
{
 long winid1, winid2;

 prefsize(img1->xsize,img1->ysize);
 winid1=winopen("grayscale view");
 RGBmode();
 singlebuffer();
 gconfig();

 prefsize(img2->xsize,img2->ysize);
 winid2=winopen("gradient view");
 RGBmode();
 singlebuffer();
 gconfig();

 qdevice(REDRAW);
 qdevice(LEFTMOUSE);
 qdevice(MIDDLEMOUSE);
 qdevice(RIGHTMOUSE);
 qdevice(ESCKEY);

 printf("\nThe following options are available . . .\n");
 printf("\n    Left mouse -> Display grayscale with remaining lines\n");
 printf("\n   Middle mouse -> Display remaining lines on white\n");
 printf("\n    Right mouse -> Display gradient view of only object\n");
 printf("\nRedraw Grayscale -> Display only grayscale image\n");
 printf("\n Redraw Gradient -> Display entire gradient\n");
 printf("\n       ESC key -> Quit program\n");

 display_all_loop(img1,winid1,img2,winid2,img3);
```

```
free(img1->imgdata.bitsptr);
free(img1);
winclose(winid1);
free(img2->imgdata.bitsptr);
free(img2);
winclose(winid2);
free(img3->imgdata.bitsptr);
free(img3);


} /* End display_all */



/*----------- ----------------------------------------------------
     FUNCTION: display_all_loop(img1,winid1,img2,winid2,img3)
      PURPOSE: Allows user to alternate between various displays for the
               2 windows id1 & id2.
   PARAMETERS:  img1 - ptr to grayscale image
                winid1 - name for window #1
                img2 - ptr to gradient image
                winid2 - name for window #2
                img3 - ptr to object only gradient image
      RETURNS: none
    CALLED BY: display_all
        CALLS: winset(name), lrectwrite(q,r,x,y,p), qread(&v),
               reshapeviewport(), draw_white_lines(p), c3f(color),
               clear(), draw_black_lines(p)
     COMMENTS: none
     ------------------------------------------------------------------*/

display_all_loop(img1,winid1,img2,winid2,img3)
   NPSIMAGE *img1, *img2, *img3;
   long winid1, winid2;
{
 static float white[3]={1.0,1.0,1.0};
 short value;
 IMG_LINE *l;

 l = Line_list_head;
 winset(winid1);
 lrectwrite(0,0,img1->xsize-1,img1->ysize-1,img1->imgdata.bitsptr);
 winset(winid2);
 lrectwrite(0,0,img3->xsize-1,img2->ysize-1,img2->imgdata.bitsptr);

 while(TRUE)
 {
 switch(qread(&value))
  {
  case REDRAW:
    winset((long)value);
```

```
     reshapeviewport();
     if(value == winid1) lrectwrite(0,0,img1->xsize-1,img1->ysize-1,img1->imgdata.bitsptr);
     if(value == winid2) lrectwrite(0,0,img2->xsize-1,img2->ysize-1,img2->imgdata.bitsptr);
     break;
   case LEFTMOUSE:
     if(value == 0)
     {
      winset(winid1);
      lrectwrite(0,0,img1->xsize-1,img1->ysize-1,img1->imgdata.bitsptr);
      draw_lines(1,White);
     }
     break;
   case MIDDLEMOUSE:
     if(value == 0)
     {
      winset(winid1);
      c3f(white);
      clear();
      draw_lines(1,Black);
     }
     break;
   case RIGHTMOUSE:
     if(value == 0)
     {
      winset(winid2);
      lrectwrite(0,0,img3->xsize-1,img3->ysize-1,img3->imgdata bitsptr);
     }
     break;
   case ESCKEY:
     if(value == 0) return;
     break;
   default:
     break;
   } /* End switch */
 } /* End while */

} /* End display_all_loop */




/*-----------------------------------------------------------------
    FUNCTION: dimensionout(object)
       PURPOSE: Print out results of object analysis.
  PARAMETERS: object - Structure which contains all info on object
       RETURNS: none
    CALLED BY: locate.c
         CALLS: none
    COMMENTS: none
   ---------------------------------------------------------------*/
```

159

```c
void dimensionout(object)
    OBJECT_DATA object;
{


  printf("\n\nRange to object is %.2f cm\n",object.range);
  printf("\n\nObject is %.2f cm  by  %.2f cm\n",object.height, object.width);


}
```

```
/*------------------------------------------------------------------
 FILENAME: locateimagesupport.h
 PURPOSE:  Provides various routines in support of object recognition
       programs.
  AUTHOR:  Mark DeClue
   DATE:  20 Aug 93

   FILES:  setup_model, process_all_image, eliminate_matching_lines,
        delete_imgline, filter_model_lines, combine_horzmodel_lines,
        endpt_test
 GLOBALS:  hmodlist, vmodlist
 COMMENTS: none


 ----------------------------------------------------------------*/
```

```c
#define White 1
#define Black 2

LINE *hmodlist=NULL, /* Globals for model linelist and model vert linelist */
    *vmodlist=NULL;
```

```
/*------------------------------------------------------------------
  FUNCTION: setup_model()
  PURPOSE: Gets desired position, sets up 5th deck database, & returns
       model view for this position.
 PARAMETERS: none
  RETURNS: pointer of type LINE_HEAD to the model line list
 CALLED BY: locate.c
   CALLS: make_world(), get_view(x,y,z,t,world,fl)
  COMMENTS: none
  -------------------------------------------------------------*/
```

```c
LINE_HEAD *setup_model()
```

```c
{
  LINE_HEAD *m;
    WORLD *fifthfloor;
    double X, Y, Z, THETA, focal_length=1.4;

  foreground();
/* ---Initial pose & max pixel difference acceptance--- */
  printf("\nEnter stopped robot position . . .\n");
  printf("\nX: ");
  scanf("\n%lf",&X);
  printf("\nY: ");
  scanf("\n%lf",&Y);
  printf("\nZ: ");
  scanf("\n%lf",&Z);
  printf("\nTHETA: ");
  scanf("\n%lf",&THETA);
  fflush(stdout);


/* ---Set up 5th deck database model and get current view--- */

  fifthfloor = make_world();
  printf("\nWorld has been generated. Now obtaining model view . . .");
  m=get_view(X,Y,Z,THETA,fifthfloor,focal_length);
  printf("View obtained\n\n ");


  return (m);

} /* ---End setup_model---*/



/*-----------------------------------------------------------------------
   FUNCTION: process_all_image(ptr1,ptr2,ptr3,ptr4)
    PURPOSE: Creates grayscale & gradient images and does line finding
 PARAMETERS: ptr1 - ptr to input rgb image
         ptr2 - ptr to grayscale image
         ptr3 - ptr to gradient image
         ptr4 - ptr to object only gradient image
    RETURNS: none
  CALLED BY: locate.c
     CALLS: rgbalong_to_bwlong(p1,&p2), pixel_membership(z,v),
         set_pixel_black(&p), set_pixel_white(&p),
         check_active_edges(), line_test(r)
  COMMENTS: Pixel inclusion based on a threshold value of 5000000 for
         gradient magnitude.

         Least Squares Fit method for line-finding from "Sonar
         Data Interpretaion for Autonomous Mobile Robots" by
         Y.Kanayama, T.Noguchi, & B.Hartman, 1990.
```

161

Function rgbalong_to_bwlong courtesy of M. Zyda.
--*/

```c
void process_all_image(ptr1,ptr2,ptr3,ptr4)
    long *ptr1, *ptr2, *ptr3, *ptr4;
{
    EDGE *reg;
    double  dx, dy, Th = 5000000.0*5000000.0;
    register int i = 0, z = 0;

/* ---Calculate bw values for first 2 rows of input image--- */
    for(i=0; i<(2*Xdim)+2; ++i)
      {
        rgbalong_to_bwlong(ptr1[i],&ptr2[i]);
      }


    for(i = Xdim + 1; i < (Xdim*(Ydim-1))-1; ++i)
      {
        /* Convert color(img1) to b/w(img2) for pixel on next row up and one
                    pixel over to the right so that all eight neighbors of pixel i
                    have black & white light intensities. */
        rgbalong_to_bwlong(ptr1[i+Xdim+1],&ptr2[i+Xdim+1]);

        /* Ensure pixel i is not in leftmost or rightmost column */
        if((i%Xdim != 0) && (i%Xdim != Xdim-1))
                    {
                    /* Calculate dx,dy via Sobel operator for pixel i. */

                    dx = (-ptr2[i+Xdim-1] + ptr2[i+Xdim+1]
                                -(2 * ptr2[i-1])+ (2 * ptr2[i+1])
                                -ptr2[i-Xdim-1] + ptr2[i-Xdim+1]);

                    dy = ( ptr2[i+Xdim-1] + (2*ptr2[i+Xdim]) + ptr2[i+Xdim+1]
                                -ptr2[i-Xdim-1] - (2*ptr2[i-Xdim]) - ptr2[i-Xdim+1]);

                    if((dx*dx)+(dy*dy) > Th)
                      {
                        pixel_membership(z,atan2(dy,dx));
                        set_pixel_black(&ptr3[z]);
            set_pixel_black(&ptr4[z]);
                      }
                    else
                      {
                        set_pixel_white(&ptr3[z]);
            set_pixel_white(&ptr4[z]);
                      }

                    ++z; /* Increment the pixel counter z for the gradient image. */
                    }
```

162

```c
        /* If pixel i is in the leftmost column, do check_active_edges(). */
        else if(i%Xdim == 0)
                        {
                          check_active_edges();
                        }

    } /* endfor i */

/* ---Check remaining EDGEs for lines--- */
 reg = Past_edge_list_head;
 while(reg != NULL)
   {
     line_test(reg);
     reg = reg->next;
   }
} /* End process_all_image */




/*-----------------------------------------------------------------
    FUNCTION: filter_model_lines()
    PURPOSE: Removes model lines which are <2 pixels in length or lie
        entirely above y coord of 486
  PARAMETERS: none
    RETURNS: y height of edge between floor and wall
  CALLED BY: locate.c
     CALLS: free(m)
  COMMENTS: none
    -------------------------------------------------------------*/

double filter_model_lines()
{
   LINE *mdh, *mdv, *modtemp;
  double flooredge=250.0, yavg;
FILE *hmodlinefile, *vmodlinefile;

  mdh = hmodlist; /* List of non-vertical model lines */
  mdv = vmodlist; /* List of vertical model lines */

hmodlinefile=fopen("hmodel_lines.text","w");
vmodlinefile=fopen("vmodel_lines.text","w");

  while(mdh != NULL)
  {
  /* First see if this edge is the lowest horz in the model */

  yavg = (mdh->Y1 + mdh->Y2)/2.0;
  if((mdh->X2 > 200.0) && (mdh->X1 < 400.0) && (yavg < flooredge))
    flooredge = yavg;
```

163

```c
/* Test non-verticals for length < 2 or location above 486 */
if(((fabs(mdh->X2 - mdh->X1) < 2.0) || ((mdh->Y1 > 486) && (mdh->Y2 > 486)))
{
 /* If removal criteria met, delete from the linked list */
 if(hmodlist == mdh)
 {
  hmodlist = mdh->NEXT;
  free(mdh);
  mdh = hmodlist;
 }
 else
 {
  modtemp->NEXT = mdh->NEXT;
  free(mdh);
  mdh = modtemp->NEXT;
 }
}

 else
 {
  fprintf(hmodlinefile,"y1: %.2f  y2: %.2f  x1: %.2f   x2: %.2f\n",
       mdh->Y1,mdh->Y2,mdh->X1,mdh->X2);
  modtemp = mdh;
  mdh = mdh->NEXT;
 }
}


while(mdv != NULL)
{
 /* Test verticals for length < 2 or location above 486 */

 if(((fabs(mdv->Y2 - mdv->Y1) < 2.0) || ((mdv->Y1 > 486) && (mdv->Y2 > 486)))
 {
  /* If removal criteria met, delete from the linked list */
  if(vmodlist == mdv)
  {
   vmodlist = mdv->NEXT;
   free(mdv);
   mdv = vmodlist;
  }
  else
  {
   modtemp->NEXT = mdv->NEXT;
   free(mdv);
   mdv = modtemp->NEXT;
  }
 }

 else
```

```c
      {
      fprintf(vmodlinefile,"y1: %.2f y2: %.2f x1: %.2f  x2: %.2f\n",
          mdv->Y1,mdv->Y2,mdv->X1,mdv->X2);

      modtemp = mdv;
      mdv = mdv->NEXT;
      }
    }

  fclose(hmodlinefile);
  fclose(vmodlinefile);

  return(flooredge);
  } /* End filter_model_lines */




/*------------------------------------------------------------------
   FUNCTION: Combine_horzmodel_lines()
    PURPOSE: Combines model line segments into contiguous lines
  PARAMETERS: none
    RETURNS: none
  CALLED BY: locate.c
     CALLS: none
   COMMENTS: updates the list pointed to by the global hmodlist
  --------------------------------------------------------------*/

void combine_horzmodel_lines()
  {
  LINE *temp, *mdh1, *mdh2;
  double ylimit=1.0, xlimit= -2.0; /*pixel dist to which line segments are
                       considered  close enough to combine */

  mdh1 = temp = hmodlist;

  while(mdh1 != NULL)
    {
    mdh2 = mdh1->NEXT;

    while(mdh2 != NULL)
      {
      if((fabs(mdh1->Y1 - mdh2->Y1) < ylimit) && /* lines are same height */
         ((mdh2->X1 - mdh1->X2) > xlimit) &&
         (mdh2->X2 < mdh1->X2))
        {
        mdh1->X2 = mdh2->X2;
        temp->NEXT = mdh2->NEXT;
        free(mdh2);
        mdh2 = temp->NEXT;
        }
```

165

```
    else if((fabs(mdh1->Y1 - mdh2->Y1) < ylimit) && /* lines are same height */
        ((mdh1->X1 - mdh2->X2) > xlimit) &&
        (mdh2->X1 > mdh1->X1))
        {
        mdh1->X1 = mdh2->X1;
        temp->NEXT = mdh2->NEXT;
        free(mdh2);
        mdh2 = temp->NEXT;
        }

    else
        {
        temp = mdh2;
        mdh2 = mdh2->NEXT;
        }
    } /*end inner while */

    mdh1 = mdh1->NEXT;
    temp = mdh1;

 } /*end outer while */

} /* end combine_horzmodel_lines */




/*--------------------------------------------------------------------
    FUNCTION: eliminate_matching_lines(pixdiff,flooredge)
    PURPOSE: Removes image lines which match to model lines
 PARAMETERS: pixdiff - user chosen value for max allowable dist between
             an image and model line (in pixels) in order to have a match.
             flooredge - y coord of wall to floor edge based on model.
    RETURNS: none
 CALLED BY: locate.c
    CALLS: atan2(x,y), free(m)
    COMMENTS: any img lines within 20 pixels of the left or right edge
             of the image or 2 pixels from bottom are also deleted.
    ------------------------------------------------------------------*/

void eliminate_matching_lines(pixdiff,flooredge)
    double pixdiff,flooredge;
{

 IMG_LINE *imgtemp, *im, *delete_imgline();
    LINE *md, *mdv;
    double horz_mid, /* avg y coord for a horizontal image line */
        diag=0.0;
     int remline=0, tmpcnt=0,
        /* flags */
```

166

```
                    remv=0, /* set to 1 if line w/in 20 pixels of left or right */
                    horz=0, vert=0, /* set to 1 if line fits horz/vert criteria */
                    endpt_test();

   imgtemp = Line_list_head; /* List of image lines */
   im = Line_list_head;

   printf("\n\nflooredge is  %.2f\n",flooredge);

   while(im != NULL)
   {


if((im->p1.y > flooredge) && (im->p2.y > flooredge))
{
  im = delete_imgline(im,imgtemp);
  remline++;
}
else
{


    md = hmodlist; /* List of non-vertical model lines */
    mdv = vmodlist; /* List of vertical model lines */

/* ---Sort out if img line is vert, horz, or diagonal and test for elimination
    based on closeness to edge of picture frame--- */

    if(fabs(im->phi) < 0.0698) /* Looking for vert lines (w/in 4 deg of vert) */
    {
      if((im->p1.x < 20) || (im->p1.x > 626)) remv=1;
      vert=1;
      im->orient=1;
      if((im->p1.y < 250.0) || (im->p2.y < 250.0))
        im->obstacle=1;
      else im->obstacle=0;  /* Any lines above 250 pixels can't effect robot */
    }
    else if((1.5708 - fabs(im->phi)) < 0.061) /* Looking for horz lines */
    {
      horz_mid=(im->p1.y + im->p2.y)/2.0; /* Calc avg y coord for image line */
      im->orient=2;
      horz=1;
      if(horz_mid < 250.0)
        im->obstacle=1;
      else im->obstacle=0;  /* Any lines above 250 pixels can't effect robot */
    }
    else
    {
      diag = atan2((im->p2.x - im->p1.x),(im->p2.y - im->p1.y));
      im->orient=3;
      horz=1;
```

```c
    if((im->p1.y < 250.0) II (im->p2.y < 250.0))
      im->obstacle=1;
    else im->obstacle=0;  /* Any lines above 250 pixels can't effect robot */
  }


/* ---Now that preliminary analysis is done on the line, compare to model,
    delete as required, and update pointers/counters--- */

  while((mdv != NULL) && (vert))
  {
    /* Remove vert img line if within pixdiff of a vert model line or if
       if it was set for removal because it was within 20 pixels of an edge */

    if((((fabs(im->p1.x - mdv->X1) < pixdiff) && endpt_test(mdv,im,pixdiff)) II remv)
    {
      remv=0;
      vert=0;
      remline ++;
      im = delete_imgline(im,imgtemp);
    }

    else mdv = mdv->NEXT;

  } /* ---End 2nd while---*/

  while((md != NULL) && (horz))
  {
    /* Remove img line if within pixdiff of a model line, it lies within 2
       pixels of bottom, or it's diagonal and is oriented within 5 degrees of
       a model diagonal line */

    if((((fabs(horz_mid - md->Y1) < pixdiff) && endpt_test(md,im,pixdiff)) II
       (horz_mid < 2) II
       (fabs(atan2((md->X2 - md->X1),(md->Y2 - md->Y1)) - fabs(diag)) < 0.0873))
    {
      horz=0;
      remline ++;
      im = delete_imgline(im,imgtemp);
    }

    else md = md->NEXT;

  } /* ---End 3rd while--- */

  if(remline == tmpcnt) /* line was not removed, so update pointers */
  {
    imgtemp = im;
    im = im->next;
  }
  else tmpcnt=remline;
```

168

```c
        vert=0;
        horz=0;
        diag=0.0;
     }
    } /* ---End 1st while--- */


    Linecount = Linecount-remline; /* Update Linecount to reflect loss of lines */
    printf("\nNumber lines remaining = %d\n",Linecount);


} /* End eliminate_matching_lines */




/*-----------------------------------------------------------------
   FUNCTION: delete_imgline
    PURPOSE: Deletes an image line from the linked list
 PARAMETERS: im - pointer to the location being removed from the
               linked list
           imtemp - pointer to list location immediately before that
                  to which im points
    RETURNS: pointer to the new current location
  CALLED BY: eliminate_matching_lines
      CALLS: none
   COMMENTS: none
-----------------------------------------------------------------*/

IMG_LINE *delete_imgline(im,imgtemp)
    IMG_LINE *im,*imgtemp;
{

   if(Line_list_head == im)
   {
    Line_list_head = im->next;
    free(im);
    im = Line_list_head;
   }
   else
   {
    imgtemp->next = im->next;
    free(im);
    im = imgtemp->next;
   }
   return(im);

} /* end delete_imgline */
```

```
/*--------------------------------------------------------------------
   FUNCTION: endpt_test(md,im,delta)
    PURPOSE: determine if the image line is within delta of the confines
          of the model line
   PARAMETERS: md - ptr to a model line
           im - ptr to an image line
           delta - max pixel difference for endpt inclusion
    RETURNS: 1 if within endpt limit / 0 if outside of limit
   CALLED BY: eliminate_matching_lines
     CALLS: none
     COMMENTS: none
   --------------------------------------------------------------*/

int endpt_test(md,im,delta)
     LINE *md;
    IMG_LINE *im;
      double delta;
{
 double mdelta;

 mdelta = -delta;
 switch(im->orient)
 {
  case 1:
/*printf("\n\n%.2f %.2f and   %.2f %.2f\n",md->Y2,im->p2.y,md->Y1,im->p1.y);*/
    if((im->p1.y > md->Y1) && (im->p2.y < md->Y2))
     return(1);
    else return(0);
    break;
   case 2:
    if(im->p1.x > im->p2.x)
    {
     if((im->p2.x > md->X2) && (im->p1.x < md->X1))
      return(1);
     else return(0);
    }
    else
    {
     if((im->p1.x > md->X2) && (im->p2.x < md->X1))
      return(1);
     else return(0);
    }
    break;
   case 3:
   default:
    return(0);
    break;
 }
} /* end endpt_test */
```

```
/*-------------------------------------------------------------------
  FILENAME: locateobjectsupport.h
  PURPOSE: Provides various routines in support of object recognition.
  AUTHOR: Mark DeClue
   DATE: 20 Aug 93

   FILES: isolate_object, iso_object, exchange, hpartition, vpartition,
       Hsort, Vsort, get_range, endpt_test, get_dimensions,
       avoid_object, fillarray
  GLOBALS: none
  COMMENTS: none

  ----------------------------------------------------------------*/


IMG_LINE harray[100], varray[100];
    int vcnt=0, hcnt=0;




/*-------------------------------------------------------------------
   FUNCTION: fillarray()
   PURPOSE: Fills varray & harray with lines from Line list.
  PARAMETERS: none
   RETURNS: none
  CALLED BY: locate.c
    CALLS: none
   COMMENTS: Note that element zero does not get data
  ----------------------------------------------------------------*/

fillarray()
{
   IMG_LINE *l = Line_list_head;
  int k=1;

  varray[vcnt].p1.x = 0.0;
  varray[vcnt].p2.x = 0.0;
  varray[vcnt].p1.y = 0.0;
  varray[vcnt].p2.y = 0.0;
  harray[hcnt].p1.x = 0.0;
  harray[hcnt].p2.x = 0.0;
  harray[hcnt].p1.y = 0.0;
  harray[hcnt].p2.y = 0.0;


  while(l!=NULL)
  {
   if(l->orient == 1)
   {
    vcnt++;
```

171

```c
      varray[vcnt].p1.x = l->p1.x;
      varray[vcnt].p2.x = l->p2.x;
      varray[vcnt].p1.y = l->p1.y;
      varray[vcnt].p2.y = l->p2.y;
      varray[vcnt].obstacle = l->obstacle;
    }
   else if(l->orient == 2)
   {
     hcnt++;
     harray[hcnt].p1.x = l->p1.x;
     harray[hcnt].p2.x = l->p2.x;
     harray[hcnt].p1.y = l->p1.y;
     harray[hcnt].p2.y = l->p2.y;
     harray[hcnt].obstacle = l->obstacle;
   }

     l = l->next;
  }

  Hsort(1,hcnt,harray);
  Vsort(1,vcnt,varray);
/*
  while(k <= hcnt)
  {
   printf("\n y = %.2f\n",harray[k].p1.y);
   k++;
  }
  k=1;
  while(k <= vcnt)
  {
   printf("\n x = %.2f\n",varray[k].p1.x);
   k++;
  }
*/
  printf("\n\n vcnt = %d  hcnt = %d\n",vcnt,hcnt);
} /* End fillarray */




/*------------------------------------------------------------------


  The following routines are used for sorting img lines into
  ascending order to allow grouping for multi-object analysis.
  The implementation is adapted from algorithms presented in
  ref [MAN89] of the thesis.

  --------------------------------------------------------------*/
```

```
exchange(A,B,V_line)
    int A,B;
    IMG_LINE V_line[];
{
 IMG_LINE temp;

 temp = V_line[A];
 V_line[A] = V_line[B];
 V_line[B] = temp;
}



int hpartition(left,right,H_line)
    int left, right;
    IMG_LINE H_line[];
{
 double pivot;
 int L,R,middle;

 pivot = H_line[left].p1.y;
 L = left;
 R = right;
 while(L < R)
  {
   while((H_line[L].p1.y <= pivot) && (L <= right))
    L=L+1;
   while((H_line[R].p1.y > pivot) && (R >= left))
    R=R-1;
   if(L < R)
    exchange(L,R,H_line);
  }
 middle = R;
 exchange(left,middle,H_line);

 return(middle);
}



int vpartition(left,right,V_line)
    int left, right;
    IMG_LINE V_line[];
{
 double pivot;
 int L,R,middle;

 pivot = V_line[left].p1.x;
 L = left;
 R = right;
 while(L < R)
```

```
    {
     while((V_line[L].p1.x <= pivot) && (L <= right))
       L=L+1;
     while((V_line[R].p1.x > pivot) && (R >= left))
       R=R-1;
     if(L < R)
       exchange(L,R,V_line);
    }
   middle = R;
   exchange(left,middle,V_line);

   return(middle);
   }



Hsort(left,right,H_line)
     IMG_LINE H_line[];
     int left, right;

{
  int middle;
  int partition();

  if(left < right)
   {
    middle = hpartition(left,right,H_line);
    Hsort(left,middle-1,H_line);
    Hsort(middle+1,right,H_line);
   }


  }



Vsort(left,right,V_line)
     IMG_LINE V_line[];
     int left, right;

{
  int middle;
  int partition();

  if(left < right)
   {
    middle = vpartition(left,right,V_line);
    Vsort(left,middle-1,V_line);
    Vsort(middle+1,right,V_line);
   }


  }
```

```
/*------------------------------------------------------------
   FUNCTION: iso_object()
   PURPOSE: Locates object in multi-object environment
 PARAMETERS: none
   RETURNS: object - Info on the position of the object
 CALLED BY: locate.c
    CALLS: horz_check()
  COMMENTS: none
 ------------------------------------------------------------*/


OBJECT_DATA iso_object(img)
    NPSIMAGE *img;


{
  OBJECT_DATA object;
    IMG_LINE *l;
       long *ptr,winid,xlen;
register int z;
       char linename[3];
        int i=1, j=2, match=0, bottom, top, horz_check();


  xlen = img->xsize;
  ptr = img->imgdata.bitsptr;
  l = Line_list_head;

  while(i <= vcnt)
  {
   if(!varray[i].obstacle) j = vcnt+1;

   while((j <= vcnt) && !match)
   {
    if(varray[j].obstacle &&
      ((varray[j].p1.x - varray[i].p1.x) > 10.0) &&
      (bottom = horz_check(varray[i].p1.x,varray[j].p1.x,
       l,hcnt,1)))
    {
     match = 1;
     top = horz_check(varray[i].p1.x,varray[j].p1.x,
               (bottom),hcnt,0);
    }
    else j++;

   } /* end inner while */
```

```
    if(!match)
    {
     i++;
     j= i+1;
    }
    else
    {
     object.left = varray[i].p1.x;
     object.right = varray[j].p1.x;
     object.bottom = harray[bottom].p1.y;
     object.top = harray[top].p1.y;
     i = vcnt + 1; /* gets out of outer while loop */
    }

  } /* end outer while */

printf("\n\nLeft= %.2f  right= %.2f  bot= %.2f  top= %.2f\n",object.left,
    object.right,object.bottom,object.top);

/* ---These 2 steps are just done during testing to 'see' how well the image
    is being isolated and isn't necessary for actual obstacle avoidance--- */

/* ---Set pixels white except those in an area around the object--- */

  for(z=xlen+1; z<((object.top+20)*xlen)+1; z++)
  {
   if(( (z%xlen) < (object.left-5)) || ((z%xlen) > (object.right+5)))
     set_pixel_white(&ptr[z]);
  }

/* ---Set remaining pixels above object to white--- */

  for(z=((object.top+20)*xlen)+1; z<(xlen*(img->ysize - 1)) - 1; z++)
  {
   set_pixel_white(&ptr[z]);
  }


  return(object);

} /* end iso_obj */




/*-------------------------------------------------------------------
   FUNCTION: horz_check(left,right,startcnt,endcnt,bottom)
   PURPOSE: Determines if a horz line exists between the two potential
          lines for the left and right side.
   PARAMETERS: left - potential left side of object
```

```
            right - potential right side of object
            startcnt - element where the array is curretly being accessed
            endcnt - last element of array with data
            bottom - 1 if looking for bottom line; 0 for top
    RETURNS: 0 if no line inbetween or element number in the array where
            the inbetween line is located
    CALLED BY: iso_object()
      CALLS: none
    COMMENTS: none
    ------------------------------------------------------------------*/

int horz_check(left,right,startcnt,endcnt,bottom)
    double left, right;
        int startcnt, endcnt, bottom;

{
  int i=startcnt+1, match=0;

  if(bottom)
  {
    while((i <= endcnt) && !match)
    {
      if(harray[i].obstacle &&
        (fabs(harray[i].p1.x - left) <= 10.0) &&
        (fabs(right - harray[i].p2.x) <= 10.0))
      match = 1;
      else i++;
    }

    if(match) return(i);
    else return(0);
  }
  else
  {
    while((i <= endcnt) && !match)
    {
      if(harray[i].obstacle &&
        ((harray[i].p1.y - harray[startcnt].p1.y) >= 10.0) &&
        (fabs(harray[i].p1.x - left) <= 10.0) &&
        (fabs(right - harray[i].p2.x) <= 10.0))
      match = 1;
      else i++;
    }

    if(match) return(i);
    else return(0);
  }

} /* end horz_check */
```

```
/*-----------------------------------------------------------------
   FUNCTION: isolate_object(img,object)
    PURPOSE: Searches the remaining img lines (hopefully an object is
             present) and whites out all but an area around the object.
             Then gets data needed by robot for obstacle avoidance.
 PARAMETERS: img - ptr to gradient object only image
             object - structure which will hold range & dimension info
    RETURNS: object - structure holding range & dimension info
  CALLED BY: locate.c
      CALLS: set_pixel_white(&p)
   COMMENTS: Original obj isolation routine; doesn't employ array struc.
             Applicable to single object case.
   -------------------------------------------------------------------*/


OBJECT_DATA isolate_object(img)
    NPSIMAGE *img;


{
    IMG_LINE *l, *leftline=NULL, *rightline=NULL;
  OBJECT_DATA object;
     double clearance, inittop;
      long *ptr,winid,xlen;
  register int z;
       int counter=0;
       char linename[3];


  xlen = img->xsize;
  ptr = img->imgdata.bitsptr;
  l = Line_list_head;


  foreground();


  leftline = (IMG_LINE *)malloc(sizeof(IMG_LINE));
  rightline = (IMG_LINE *)malloc(sizeof(IMG_LINE));
  leftline->p1.x = 0.0;
  leftline->p1.y = 0.0;
  leftline->p2.x = 0.0;
  leftline->p2.y = 0.0;
  rightline->p1.x = 0.0;
  rightline->p1.y = 0.0;
  rightline->p2.x = 0.0;
  rightline->p2.y = 0.0;


/* ---From remaining lines, find vert/obstacle line furthest to left--- */

  object.left = 626.0;
```

```c
while(l != NULL)
{
  /* orient=1 means vertical and obstacle means y<250 */
  if((l->orient==1) && l->obstacle && (l->p1.x < object.left))
  {
    object.left = l->p1.x;
    leftline = l;  /* This line is the left edge of the object */
  }
  l = l->next;
} /*end while*/

l = Line_list_head;

/* ---From remaining lines, find vert/obstacle line second furthest to left
     but at least 5 pixels away from line set as left side--- */

object.right = 626.0;
while(l != NULL)
{
  if((l->orient==1) && l->obstacle && (l->p1.x < object.right) &&
                    (l->p1.x > (object.left+5.0)))
  {
    object.right = l->p1.x;
    rightline = l;  /* This line is the right edge of the object */
  }
  l = l->next;
} /*end while*/

  printf("\nLeftside = %.2f  Rightside = %.2f\n",object.left,object.right);

/*
printf("\n\nleftp1 = %.2f , %.2f\n",leftline->p1.x,leftline->p1.y);
printf("\n\nrightp1 = %.2f , %.2f\n",rightline->p1.x,rightline->p1.y);
printf("\n\nleftp2 = %.2f , %.2f\n",leftline->p2.x,leftline->p2.y);
printf("\n\nrightp2 = %.2f , %.2f\n",rightline->p2.x,rightline->p2.y);
*/
/* ---Determine top/bottom most point for the sides of the object--- */

  if(leftline->p1.y >= leftline->p2.y)
  {
    inittop = leftline->p1.y;
    object.bottom = leftline->p2.y;
  }
  else
  {
    inittop = leftline->p2.y;
    object.bottom = leftline->p1.y;
  }

  if(rightline->p1.y >= rightline->p2.y)
  {
```

```
      if(rightline->p1.y >= inittop) inittop = rightline->p1.y;
      else if(rightline->p2.y <= object.bottom) object.bottom = rightline->p2.y;
    }
    else
    {
      if(rightline->p2.y >= inittop) inittop = rightline->p2.y;
      else if(rightline->p1.y <= object.bottom) object.bottom = rightline->p1.y;
    }

printf("\n\ninittop = %.2f    initbottom = %.2f\n\n",inittop,object.bottom);

/* ---Search for the image lines which make up the top and bottom edges--- */

    l = Line_list_head;
    object.top = 250.0;

    while(l != NULL)
    {
      if((l->orient == 2) && ((fabs(object.left - l->p1.x)) <= 10.0) &&
        ((fabs(object.right - l->p2.x)) <= 10.0)) /* Line is within 10 pixels
                                of left/right edge */
      {
        if(l->p1.y <= object.bottom) object.bottom = l->p1.y;
        if((l->p1.y > inittop) && (l->p1.y <= object.top)) object.top = l->p1.y;
      }
      l = l->next;
    }

printf("\nBottom at %.2f    Top at %.2f\n",object.bottom,object.top);

/* ---These 2 steps are just done during testing to 'see' how well the image
       is being isolated and isn't necessary for actual obstacle avoidance--- */

/* ---Set pixels white except those in an area around the object--- */

    for(z=xlen+1; z<((object.top+20)*xlen)+1; z++)
    {
      if(( (z%xlen) < (object.left-5)) || ((z%xlen) > (object.right+5)))
        set_pixel_white(&ptr[z]);
    }

/* ---Set remaining pixels above object to white--- */

    for(z=((object.top+20)*xlen)+1; z<(xlen*(img->ysize - 1)) - 1; z++)
    {
      set_pixel_white(&ptr[z]);
    }

    return(object);

} /* End isolate_object */
```

```
/*----------------------------------  ----------------------------------
    FUNCTION: get_range
    PURPOSE: Determines range to an object by base pixel height
 PARAMETERS: bottom - the vert coord in pixels of the object's base
    RETURNS: range - the range to the object
  CALLED BY: locate.c
      CALLS: none
   COMMENTS: Uses range equation as derived in thesis
 ----------------------------------------------------------------------*/

OBJECT_DATA get_range(object)
 OBJECT_DATA object;
{

  double alpha=0.032, h=86.36, delta=8.26E-4;
  double arg, pix;

  pix = (486.0/2.0) - object.bottom;
  arg = alpha+pix*delta;
  object.range = h / tan(arg);

  return(object);

} /* end get_range */




/*--------------------------------------------------------------------
    FUNCTION: get_dimensions
    PURPOSE: Determines the dimensions of an object
 PARAMETERS: range - the range to the object
            object - the structure which upon being sent to this routine
                     contains the left/right sides in pixels along the
                     horz axis and top/bottom along the vert axis
    RETURNS: object - the same left/right, top/bottom values as sent in
                     but also the calculated width and height in cm.
  CALLED BY: locate.c
      CALLS: none
   COMMENTS: Uses image pixel to physical length conversion factor (delta)
            of 1205/d as described in thesis
 ----------------------------------------------------------------------*/

OBJECT_DATA get_dimensions(object)
 OBJECT_DATA object;
{
  double pixel_width, pixel_height, delta;
```

181

```
delta = 1205.0/object.range;

pixel_width = object.right - object.left;
pixel_height = object.top - object.bottom;

object.height = pixel_height * (1.0/delta);
object.width = pixel_width * (1.0/delta);

return(object);

} /* end get_dimensions */



/*------------------------------------------------------------------
   FUNCTION: avoid_object()
   PURPOSE: Calculates the shift distance necessary to avoid the given
         object by 1 meter
  PARAMETERS: leftside - left edge of the object to be avoided
              rightside - right edge of the object to be avoided
      RETURNS: shift - the distance (in cm) necessary to avoid object where
                  a positive value is a right shift, a negative value
                  is a left shift, and 0.0 is maintain current path.
   CALLED BY: locate.c
     CALLS: none
     COMMENTS: none
   --------------------------------------------------------------------*/

double avoid_object(leftside,rightside)
     double leftside,rightside;
{
  double deltaright, deltaleft, shift=1000.0, middle=646.0/2.0;

  deltaright=middle-rightside;
  deltaleft=leftside-middle;

  if(deltaright >= 0.0) /* obj completely to left */
  {
    if(deltaright >= 100.0) shift = 0.0;
    else shift = 100.0 - deltaright;
  }
  else if(leftside >= 0.0) /* obj completely to right */
     {
       if(deltaleft >= 100.0) shift = 0.0;
       else shift = deltaleft - 100.0;
     }
  else
  {
    if((fabs(deltaleft)) < (fabs(deltaright)))
       shift = deltaleft - 100.0;  /* obj more to right so shift left */
```

182

```c
     else shift = 100.0 - deltaright; /* obj more to left so shift right */
    }

    if((shift > 0.0) && (shift < 1000.0))
      printf("\n\nIn order to avoid the object, shift %.1f cm to right\n",shift);
    else if(shift < 0.0)
        printf("\n\nIn order to avoid the object, shift %.1f cm to left\n",-shift);
    else if(shift == 0.0)
        printf("\n\nNo path alteration necessary to avoid object\n");
    if(shift == 1000.0) /* None of the conditions applied */
    {
      shift = 0.0;
      printf("\n\nCan't interpret object; maintaining present path\n");
    }

    return(shift);
} /* end avoid_object */
```

APPENDIX C - SUPPORT ROUTINES

SHOWMODEL: This routine is simple in nature and yet proved to be very useful during much of the research work. When called, the user is prompted for x, y, z, and θ coordinates of the camera based on the original model coordinate system with distances in inches (a version which accepts data based on the current coordinate system with distances in centimeters was also created SHOW_MODEL_CM). The result of this input is a window, which may be placed anywhere on the screen, displaying the two-dimensional view generated from the wire-frame model for the input position placed over a white background.

```
/*******************************************************/
/* FUCNTION:  SHOWMODEL.C                              */
/* PURPOSE:   Presents model view based on keyboard input of  */
/*           position as X, Y, Z and theta.            */
/* PARAMETERS: none                                    */
/* RETURNS:   none                                     */
/* CALLED BY: none                                     */
/* CALLS TO:  make_world, get_view, draw_lines, free_lines   */
/* GLOBALS:   none                                     */
/*******************************************************/

#include <gl.h>              /* SiliconGraphics (r) graphic library */
#include <gl/image.h>        /* SGI image structure library         */
#include <device.h>          /* Machine-dependent device library    */
                             /* for keys and mouse-buttons          */
#include <stdio.h>           /* C standard i/o library              */
#include <math.h>            /* C math library for atan2()          */

#include "model2d+.h"            /* Wireframe model graphics code    */
#include "model5th.h"           /*          "                       */
#include "modelgraphics.h"      /*          "                       */
#include "modelvisibility.h"    /*          "                       */

#include "showmod_support.h"    /* Support structures/routines       */

#define focalength 1.4          /* Focal length                      */

main()


{
WORLD       *FifthFloor;
LINE_HEAD  *model = NULL;
long        winid;
static float  white[3] = {1.0, 1.0, 1.0};
VIEWPT      *getview;

foreground();

getview = (VIEWPT *) malloc(sizeof(VIEWPT));

/*---Get pose values for model image,
    Initialize world database for fifth floor of Spanagle Hall & determine
    2D view of environment--- */

model = setup_model();
/* ---Set up display configuration--- */

prefsize(648, 486);
```

184

```
winid = winopen("MODEL VIEW");
RGBmode();
singlebuffer();
gconfig();
winset(winid);

/* ---Display image on white background--- */

c3f(white);
clear();
draw_lines(m1->LINE_LIST, m1->VLINE_LIST);

printf("\nEnter 1 to quit -> ");
scanf("\n%lf", &getview->Z);

/* ---Wrap up viewing--- */

free_lines(m1);
winclose(winid);

printf("\n Thank's for using showmodel \n\n");

}/* ---End showmodel--- */
```

CHANGEOVERLAY: The primary use for this program is in determining if an image, based on a physical position for the camera, coincides properly with the view predicted by the model. This routine is called with an input argument of a sgi/rgb stored image. The image is then displayed in a window, which may be placed anywhere on the screen, and the user is then prompted for position input as in SHOW_MODEL. Following processing, the two-dimensional model view for this position is super-imposed over the image. A menu of options is provided next which allows any of the following to be carried out:

● Display only the image

● Display the image with the model superimposed

● Input a new position and display

```
/*****************************************************************/
/* FUNCTION:  CHANGEOVERLAY.C                                 */
/* PURPOSE:   Displays input image with wire_frame model from pose  */
/*            superimposed. Mouse buttons function as follows:     */
/*            Left mouse --- Image only                          */
/*            Middle mouse - Model lines over image              */
/*            Right mouse -- Accepts pose and displays image with  */
/*                  wire-frame model superimposed.              */
/* PARAMETERS: Stored rgb image as <image name>.pic             */
/* RETURNS:   none                                              */
/* CALLED BY:  none                                             */
/* CALLS TO:  make_world, get_view, read_sgi_rgbimage, draw_white_  */
/*            model_lines                                       */
/* GLOBALS:   CAMERA_HEIGHT, FOCAL_LENGTH (Both are constants)  */
/* COMMENTS:  Mark DeClue, 24 Feb 93                           */
/*****************************************************************/

#include <gl.h>            /* SiliconGraphics (r) graphic library */
#include <gl/image.h>      /* SGI image structure library         */
#include <device.h>        /* Machine-dependent device library    */
#include <stdio.h>         /* C standard i/o library              */
```

185

```c
#include <math.h>              /* C math library for atan2()            */
#include "2d+.h"
#include "5th.h"
#include "graphics.h"
#include "visibility.h"

#include "image_types.h"           /* Type definitions for NPSIMAGE, etc.    */
#include "match_types.h"           /* Type definitions for EDGE, IMG_LINE..  */
#include "npsimagesupport.h"       /* Some NPSIMAGE functions                */
#include "edgesupport.h"           /* EDGE and IMG_LINE building functions   */
#include "vertsupport.h"           /* Vertical EDGE and IMG_LINE supplement  */
#include "matchsupport.h"          /* LINE and IMG_LINE matching routines    */
#include "matchdisplaysupport.h"   /* Graphics display functions             */
#include "marksubs.h"              /* VIEWPT data structure                  */

main(argc, argv)
   int         argc;
   char        *argv[];
{
   WORLD        *fifthfloor;
   NPSIMAGE     *img;
   LINE_HEAD    *m = NULL;
   VIEWPT       *getpos;
   short        value;
   long         winid;
   int          keepon = 1;

foreground();

/* ---Read in the image to be displayed--- */

img = read_sgi_rgbimage(argv[1]);
printf("\n%s has been stored\n", img->name);


/* ---Set up 5th deck database model--- */

fifthfloor = make_world();

/* ---Initialize for display and control--- */

prefsize(img->xsize, img->ysize);   /* preferred size for window         */
winid = winopen(img->name);         /* open the window                   */
RGBmode();                          /* set RGBmode, singlebuffer, and    */
singlebuffer();                     /* configure the window              */
gconfig();
qdevice(REDRAW);
qdevice(RIGHTMOUSE);
qdevice(ESCKEY);
qdevice(MIDDLEMOUSE);
qdevice(LEFTMOUSE);
getpos = (VIEWPT *) malloc(sizeof(VIEWPT));

/* ---Initial display of image--- */

winset(winid);
lrectwrite(0, 0, img->xsize - 1, img->ysize - 1, img->imgdata.bitsptr);

/* ---Initial pose acceptance and display--- */

printf("\nEnter desired initial view:\n");
```

186

```c
fflush(stdout);
printf("X: ");
fflush(stdout);
scanf("\n%lf", &getpos->X);
printf("\nY: ");
scanf("\n%lf", &getpos->Y);
printf("\nZ: ");
scanf("\n%lf", &getpos->Z);
printf("\nTHETA: ");
scanf("\n%lf", &getpos->THETA);

m = get_view(getpos->X, getpos->Y, getpos->Z, getpos->THETA, fifthfloor,
    FOCAL_LENGTH);

draw_white_model_lines(m->VLINE_LIST);
draw_white_model_lines(m->LINE_LIST);

/* ---List available options--- */

printf("\nOptions available are as follows ...\n");
printf("\n    Left mouse button -> Display image only\n");
printf("  Middle mouse button -> Superimpose wire-frame model based on ");
printf("current pose\n");
printf("  Right mouse button -> Accepts new pose. Displays image & wire-");
printf("frame model\n");
printf("           ESC key -> Exit program\n");

/* ---Loop until a mouse button is pressed--- */

while (keepon == 1)
{
  switch (qread(&value))
  {
    case REDRAW:
      winset((long) value);
      reshapeviewport();
      if (value == winid)
      {
        lrectwrite(0, 0, img->xsize - 1, img->ysize - 1, img->imgdata.bitsptr);
      }
      break;
    case RIGHTMOUSE:
      if (value == 0)
      {

/* ---Get pose values for model image--- */

        printf("\nEnter desired new view:\n");
        fflush(stdout);
        printf("X: ");
        fflush(stdout);
        scanf("\n%lf", &getpos->X);
        printf("\nY: ");
        scanf("\n%lf", &getpos->Y);
        printf("\nZ: ");
        scanf("\n%lf", &getpos->Z);
        printf("\nTHETA: ");
        scanf("\n%lf", &getpos->THETA);

        m = get_view(getpos->X, getpos->Y, getpos->Z, getpos->THETA, fifthfloor,
                    FOCAL_LENGTH);
```

```
         lrectwrite(0, 0, img->xsize - 1, img->ysize - 1, img->imgdata.bitsptr);
         draw_white_model_lines(m->VLINE_LIST);
         draw_white_model_lines(m->LINE_LIST);
         printf("\nNew wire-frame image has been superimposed\n");
      }
      break;
    case ESCKEY:
      keepon = 0;
      break;
    case LEFTMOUSE:
      if (value == 0)
        lrectwrite(0, 0, img->xsize - 1, img->ysize - 1, img->imgdata.bitsptr);
      break;
    case MIDDLEMOUSE:
      if (value == 0)
      {
        draw_white_model_lines(m->VLINE_LIST);
        draw_white_model_lines(m->LINE_LIST);
      }
      break;
    default:
      break;
}/* end switch */
}/* end while */

/* ---Clean up from display work--- */

free(img->imgdata.bitsptr);    /* delete the bitmap for the image    */
free(img);                     /* delete the NPSIMAGE structure      */
winclose(winid);               /* close the window                   */

printf("\nThanks for using changeoverlay. Have a nice day!\n");
}
```

LIST OF REFERENCES

[KAH90] Kahn, P., Kitchen, L., and Riseman, E.M., *A Fast Line Finder for Vision-Guided Robot Navigation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 11, pp. 1088-1102, November 1990.

[KAN93] Kanayama, Yutaka, *Lecture Notes: CS4313, Advanced Robotics*, Naval Postgraduate School, Monterey, California, April 1993.

[MAC93] MacPherson, David L., *Automated Cartography by an Autonomous Mobile Robot*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, September 1993.

[MAN89] Manber, Udi, *Introduction to Algorithms-A Creative Approach*, Addison-Wesley Publishing Co, 1989.

[PET92] Peterson, Kevin L., *Visual Navigation for an Autonomous Mobile Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1992.

[STE92] Stein, James E., *Modelling, Visibility Testing and Projection of an Orthogonal Three Dimensional World in Support of a Single Camera Vision System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1992.

[ULL91] Ullman, Shimon and Basri, Ronen, *Recognition by Linear Combinations of Models*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 13, No. 10, pp 992-1005, October 1991.

BIBLIOGRAPHY

Ballard, Dana H. and Brown, Christopher M., *Computer Vision*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

Grimson, W. Eric L., *Object Recognition by Computer: The Role of Geometric Constraints*, The Mit Press, Cambridge, Massachusetts, 1990.

Kanayama, Yutaka, *Yamabico User's Manual*, Naval Postgraduate School, Monterey, California, April 1993.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center   2
Cameron Station
Alexandria, VA   22304-6145

Dudley Knox Library   2
Code 052
Naval Postgraduate School
Monterey, CA   93943-5002

Chairman, Code CS   2
Computer Science Department
Naval Postgraduate School
Monterey, CA   93943

Dr Yutaka Kanayama, Code CS/KA   2
Computer Science Department
Naval Postgraduate School
Monterey, CA   93943

Don Brutzman, Code OR/BR   1
Naval Postgraduate School
Monterey, CA   93943

Lt Mark J. DeClue   1
145 St Croix Ave
Cocoa Beach, FL 32931